ioProgrammo Anno XI - N°9 (118)

ANTEPRIMA OPENXML: IL NUOVO FORMATO ALLA BASE DI OFFICE 2007. ECCO COME CREARE UN NOSTRO WORD PERSONALIZZATO

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90

VERSIONE STANDARD RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • SETTEMBRE 2007 • ANNO XI, N.9 (118)

GRANDE FRATELLO con il GPS

Rintraccia chi vuoi tu fornendogli solo un cellulare e poco altro

- ✔ **SVILUPPA** un'applicazione per il telefonino che si interfaccia con il GPS
- ✔ **TRASMETTI** la posizione geografica ad un web services
- ✔ **MEMORIZZA** i dati su un DB qualunque senza preoccuparti della struttura del server
- ✔ **INVIA** tutto ad una mappa di Google per visualizzare la traccia

IL CODICE COMPLETO NEL CD

DAL WEB AD EXCEL CON UN CLICK

Preleva le informazioni dalla tua Intranet e crea report sofisticati utilizzando una libreria semplice e gratuita

INTERNET

DOCUMENTI FUORI DI ROOT

Proteggi i download. Tieni i file fuori dal server e spostali solo quando necessario

SECURITY

NEL WEB ENTRA SOLO CHI DICO IO

Con LiveUser e PHP gestiamo autenticazione e ruoli degli utenti in modo rapido

.NET

GOOGLE DESKTOP FATTO IN CASA

Usa la reflection per creare un'applicazione modulare che indicizza i contenuti dell'HD

.NET

I PROTOCOLLI DI INTERNET

La guida completa per gestire tutti i servizi della rete dal tuo codice: mail, news, ftp...

APPUNTAMENTI GESTITI CON WCF

Ecco gli esempi per lavorare subito con il nuovo Windows Communication Foundation

JAVA

AJAX FACILE CON THINWIRE

Applicazioni Web uguali a quelle per il Desktop con il framework che rende tutto estremamente semplice

C++

CODICE PER TUTTI CON LE WXWIDGETS

Programmiamo un'applicazione grafica che funziona con Linux e Windows senza modifiche

PATTERN

IL PATTERN COMMAND

Ecco come implementare il sistema di "redo/undo" perfetto!

2 NUOVI VIDEO

SVILUPPO PER DISPOSITIVI MOBILI
- Utilizzo dei CAB nel deployment
- Costruzione di un servizio in un dispositivo PPC

CORSI BASE

SMARTPHONE

Variabili e strutture principali. Disegniamo la nostra prima Form

UML

Diagrammi di attività: gli elementi per progettare il flusso dell'applicazione

JAVA SERVER FACES

Massimo riutilizzo con i componenti

CRIVELLO QUADRATICO DUE PAROLE "DIFFICILI" PER INDICARE METODI NECESSARI PER CIFRARE/DECIFRARE I DATI CON RSA. ECCO COSA C'È DIETRO...

GRANDE FRATELLO CON IL GPS • DAL WEB AD EXCEL CON UN CLICK • DOCUMENTI FUORI DI ROOT • NEL WEB ENTRA SOLO CHI DICO IO

EDIZIONI MASTER

24-07-2007 18:01:11

Anno XI - N.ro 09 (118) - Settembre 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, D. De Michelis, F. Grimaldi, E. Viale, V. Vessia, A. Pelleriti, F. Smelzo, C Scuderi, G. Malaga, F. Fortino, V. Arconzo
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: IOPROGRAMMO (11 NUMERI) €59,90 SCONTO 21% SUL PREZZO DI COPERTINA DI €75,90 · IOPROGRAMMO CON LIBRO (11 NUMERI) €75,90 SCONTO 30% SUL PREZZO DI COPERTINA DI €108,90 OFFERTE VALIDE FINO AL 30/09/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano
Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno

Servizio Abbonati:
☎ tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Agosto 2007

1 Anno di Computer Bild 2006, 1 Anno di Io Programmo in DVD 2006, 1 Anno di Linux Magazine in DVD 2006, 1 Anno di Office Magazine 2006, 1 Anno di Win Magazine in DVD 2006, 360 Experience, Audio/Video/Foto Bild Italia, Auto Interactive, Calcio & Scommesse, Carlo Verdone Collection, Computer Bild Italia, Computer Games Gold, Digital Japan Magazine, Digital Music, DVD Magazine, DVD Magazine Films, Family DVD Games, Filmteca in DVD, Frank Sinatra Collection, Fred Astaire Collection, Futurama Collection, Go!OnLine Internet Magazine, Home Entertainment, Horror Mania, I Classici del Cinema Musical, I DVD di Quale Computer, I DVD di Win Magazine, I DVD de La Mia Barca, I Film di Idea Web, I Filmissimi in DVD, I Film di DVD Magazine, I Gadget de La Mia Barca, I Grandi Giochi per Pc, I Libri di Quale Computer, I Mitici all'Italiana, Idea Web, Idea Web Film, InDVD, IoProgrammo, I Tecnoplus di Win Magazine, Japan Cartoon, Jerry Lewis Collection, La mia Barca, La mia Videoteca, Linux Magazine, Miami Vice in DVD, Office Magazine, Play Generation, Play Generation Games, Play Generation Plus, Play Generation Guide e Trucchi, PC Junior, Quale Computer, Softline Software World, Sport Life, Star in DVD, Video Film Collection, Win Junior, Win Magazine Giochi, Win Magazine, Win Magazine Digital Home, Yu-Gi-Oh Collection, Le Collection.

Questo mese su ioProgrammo

RITMI SERRATI

Non tutti hanno digerito ancora le nuove caratteristiche di Windows Vista che siamo già qui a parlare del suo erede: Windows 7 che dovrebbe vedere la luce nel 2010. Non che la cosa ci dispiaccia, come appassionati di tecnologia e come programmatori l'evoluzione serrata dell'informatica non può che farci piacere. Eppure leggo e ascolto tanti sviluppatori che si lamentano. L'obiezione più comune è: "lavoro tutto il giorno, dove trovo il tempo di aggiornarmi?", oppure: "la mia azienda non ha davvero bisogno di tutto questo". In alcuni casi ho la sensazione che ci sia una volontà di tirare il freno. E sebbene questa necessità di rallentare sia persino comprensibile, non mi sento di condividerla. Bisogna distinguere fra il ritmo dell'innovazione e il ritmo dell'adeguamento. è doveroso per un programmatore: "conoscere". Non per questo è altrettanto utile mettere subito in pratica. E' importante però essere preparati, la conoscenza di una nuova tecnologia può risolvere un problema ad un cliente in modo migliore e più rapido rispetto al passato. Il non conoscerla significa precludersi la possibilità di offrire un servizio migliore. Si certo, i tempi sono stretti, un programmatore che passa diverse ore al giorno seduto davanti alla sua postazione raramente troverà spazio per l'aggiornamento. Eppure bisogna compiere uno sforzo nel trovare anche quel pochissimo tempo che ci consente di leggere di sfuggita una news, un articolo tecnico o qualunque altra informazione che ci consenta di stare al passo con i tempi. Non dobbiamo lasciare che i ritmi quotidiani schiaccino la nostra curiosità e la nostra passione, che sono poi anche i motivi per cui siamo sviluppatori e non semplici fruitori della tecnologia

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

SVILUPPA un'applicazione per il telefonino e che si interfaccia con il GPS

TRASMETTI la posizione geografica ad un web services

MEMORIZZA i dati su un DB qualunque senza preoccuparsi della struttura del server

INVIA tutto ad una mappa di google per visualizzare la traccia

pag. 14

DAL WEB AD EXCEL CON UN CLICK

Preleva le informazioni dalla tua intranet e crea report sofisticati utilizzando una libreria semplice e gratuita

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto.

http://forum.ioprogrammo.it

Anno XI - N.ro 09 (118) - Settembre 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, D. De Michelis, F. Grimaldi, E. Viale, V. Vessia, A. Pelleriti, F. Smelzo, C Scuderi, G. Malaga, F. Fortino, V. Arconzo
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: IOPROGRAMMO (11 NUMERI) €59,90 SCONTO 21% SUL PREZZO DI COPERTINA DI €75,90 · IOPROGRAMMO CON LIBRO (11 NUMERI) €75,90 SCONTO 30% SUL PREZZO DI COPERTINA DI €108,90 OFFERTE VALIDE FINO AL 30/09/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano
Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno

Servizio Abbonati:
☎ **tel. 02 831212**
@ **e-mail: servizioabbonati@edmaster.it**

Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Agosto 2007

Questo mese su ioProgrammo

▼ RITMI SERRATI

Non tutti hanno digerito ancora le nuove caratteristiche di Windows Vista che siamo già qui a parlare del suo erede: Windows 7 che dovrebbe vedere la luce nel 2010. Non che la cosa ci dispiaccia, come appassionati di tecnologia e come programmatori l'evoluzione serrata dell'informatica non può che farci piacere. Eppure leggo e ascolto tanti sviluppatori che si lamentano. L'obiezione più comune è: "lavoro tutto il giorno, dove trovo il tempo di aggiornarmi?", oppure: "la mia azienda non ha davvero bisogno di tutto questo". In alcuni casi ho la sensazione che ci sia una volontà di tirare il freno. E sebbene questa necessità di rallentare sia persino comprensibile, non mi sento di condividerla. Bisogna distinguere fra il ritmo dell'innovazione e il ritmo dell'adeguamento. È doveroso per un programmatore: "conoscere". Non per questo è altrettanto utile mettere subito in pratica. E' importante però essere preparati, la conoscenza di una nuova tecnologia può risolvere un problema ad un cliente in modo migliore e più rapido rispetto al passato. Il non conoscerla significa precludersi la possibilità di offrire un servizio migliore. Si certo, i tempi sono stretti, un programmatore che passa diverse ore al giorno seduto davanti alla sua postazione raramente troverà spazio per l'aggiornamento. Eppure bisogna compiere uno sforzo nel trovare anche quel pochissimo tempo che ci consente di leggere di sfuggita una news, un articolo tecnico o qualunque altra informazione che ci consenta di stare al passo con i tempi. Non dobbiamo lasciare che i ritmi quotidiani schiaccino la nostra curiosità e la nostra passione, che sono poi anche i motivi per cui siamo sviluppatori e non semplici fruitori della tecnologia

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

GRANDE FRATELLO con il GPS

Rintraccia chi vuoi tu fornendogli solo un cellulare e poco altro

SVILUPPA un'applicazione per il telefonino che si interfaccia con il GPS

TRASMETTI la posizione geografica ad un web services

MEMORIZZA i dati su un DB qualunque senza preoccuparti della struttura del server

INVIA tutto ad una mappa di Google per visualizzare la traccia

pag. 14

RIVISTA + LIBRO + CD-ROM in edicola

I contenuti del libro

JAVASCRIPT BY EXAMPLE

Si identifica con il termine web 2.0 quell'insieme di tecnologie che fanno sì che una pagina Web sia fruibile con le stesse comodità di un'applicazione desktop. Ci riferiamo ad esempio all'aggiornamento di singole porzioni di una pagina html indipendentemente dal reload dell'intera pagina. Alla base di queste tecniche che stanno rapidamente rivoluzionando il modo di intendere il Web c'è Javascript. Si tratta di un linguaggio esistente e ben conosciuto da moltissimi anni, ma che con l'avvento delle nuove tecnologie sta conoscendo una nuova vita. È in questa ottica che viene analizzato JavaScript all'interno di questo Handbook. Abbiamo preferito proporre un numero elevato di script preconfezionati sia per consentirvi di utilizzarli all'interno delle vostre pagine Web, ma anche perché attraverso l'uso dei commenti sia possibile apprendere facilmente le tecniche che stanno alla base del Web 2.0. Lo scopo finale è quello di fornire un riferimento rapido per i problemi più immediati che sia anche una base a lungo termine per successive evoluzioni

OLTRE 90 ESEMPI COMMENTATI PER IMPARARE RAPIDAMENTE AD USARE IL LINGUAGGIO DEL WEB 2.0

- **Introduzione al linguaggio e alla sintassi**
- **Gestione del DOM html e degli stili**
- **Ottimizzazione ed efficienza di Javascript**

RIVISTA + CD-ROM in edicola

JAVA SE

Development Kit 6U2

Il compilatore indispensabile per programmare in Java

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

Come usare l'interfaccia del CDRom

IL SOFTWARE
Una accurata recensione dei contenuti

IN EVIDENZA
Il top software del mese individuato dalla redazione

IL SOFTWARE
Il software diviso in categorie per una comoda consultazione

HOME
Torna alla pagina iniziale del CD-ROM

CONTATTACI
Vuoi inviare una email alla redazione con le tue richieste

TOP SITES
I siti più interessanti del mese selezionate per te

RICERCA SOFTWARE
Il database di tutti i software pubblicati da ioProgrammo anche gli arretrati

IL SOFTWARE
L'elenco del software contenuto nelle categorie

DIMENSIONE
La dimensione del software sul CD

SALVA
Clicca qui per installare o salvare il software sul tuo PC

INFO
Abbonamenti informazioni e servizi utili

GLI ALLEGATI DI IOPROGRAMMO

Questo mese due grandi video per chi ama sviluppare per il Mobile

Sviluppo per dispositivi mobili

Utilizzo dei CAB nel deployment

Spesso, dopo aver costruito un'applicazione importante, abbiamo la necessità di effettuare un deployment altrettanto complesso che porti il dispositivo target a una configurazione finale che includa impostazioni particolari sul registry, sul file system o che permetta di visualizzare degli "shortcut" per l'avvio veloce del programma. In questo webcast verrà costruita la dll setup.dll da includere all'interno di un file CAB per implementare delle operazioni anche complesse.

Sviluppo per dispositivi mobili

Costruzione di un servizio in un dispositivo PPC

Forse non tutti sanno che sui dispositivi Pocket PC è possibile scrivere dei servizi che permettono di implementare alcune funzionalità importanti all'avvio del dispositivo. In questo Webcast vedremo come realizzarne uno e come sfruttare le opportunità che ci vengono offerte dal particolare processo Service.exe.

PERCORSI

msdn WEBCAST

Sviluppo per dispositivi mobili - Telefonare e inviare SMS utilizzando TAPI ● Sviluppo per dispositivi mobili - Gestione della batteria tramite notifiche in Windows Mobile ● Sviluppo per dispositivi mobili - Scrivere applicazioni che utilizzano Mobile GPS ● Sviluppo per dispositivi mobili - Creazione di una chat con Bluetooth

per maggior informazioni visita: http://www.microsoft.com/italy/msdn/risorsemsdn/mobile/path/mobile.mspx

FAQ

Cosa sono i Webcast MSDN?

MSDN propone agli sviluppatori una serie di eventi gratuiti online e interattivi che approfondiscono le principali tematiche relative allo sviluppo di applicazioni su tecnologia Microsoft. Questa serie di "corsi" sono noti con il nome di Webcast MSDN

Come è composto tipicamente un Webcast?

Normalmente vengono illustrate una serie di Slide commentate da un relatore. A supporto di queste presentazioni vengono inserite delle Demo in presa diretta che mostrano dal vivo come usare gli strumenti oggetto del Webcast

Come mai trovo riferimenti a chat o a strumenti che non ho disponibili nei Webcast allegati alla rivista?

La natura dei Webcast è quella di essere seguiti OnLine in tempo reale. Durante queste presentazioni in diretta vengono utilizzati strumenti molto simili a quelli della formazione a distanza. In questa ottica è possibile porre domande in presa diretta al relatore oppure partecipare a sondaggi etc. I Webcast riprodotti nel CD di ioProgrammo, pur non perdendo nessun contenuto informativo, per la natura asincrona del supporto non possono godere dell'interazione diretta con il relatore.

Come mai trovo i Webcast su ioProgrammo

Come sempre ioProgrammo cerca di fornire un servizio ai programmatori italiani. Abbiamo pensato che poter usufruire dei Webcast MSDN direttamente da CD rappresentasse un ottimo modo di formarsi comodamente a casa e nei tempi desiderati. Lo scopo tanto di ioProgrammo, quanto di Microsoft è infatti quello di supportare la comunità dei programmatori italiani con tutti gli strumenti possibili.

Su ioProgrammo troverò tutti Webcast di Microsoft?

Ne troverai sicuramente una buona parte, tuttavia per loro natura i webcast di Microsoft vengono diffusi anche OnLine e possono essere seguiti previa iscrizione. L'indirizzo per saperne di più è: www.microsoft.it/msdn/webcast. segnalo nei tuoi bookmark. Non puoi mancare.

L'iniziativa sarà ripetuta sui prossimi numeri?

Sicuramente si.

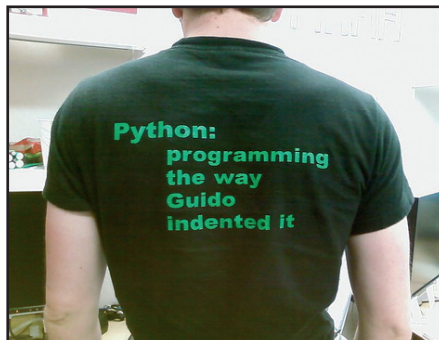
News

È ANCORA POLEMICA PER GOOGLE VIDEO

Pomo della discordia è questa volta il recente film di Harry Potter. Già più di una volta Google Video era stata accusata di ospitare sui propri sistemi video protetti da copyright. Questa volta è toccato al popolarissimo Harry Potter, recentemente apparso nelle sale di mezzo e mondo e già campione di incassi. Una presenza del genere non poteva ovviamente passare inosservata. L'accusa proviene dalla National Legal e Policy Center della Virginia, che ha pubblicato una lista dei cinquanta file più richiesti su Google Video, nella lista in questione comparirebbero video di una certa rilevanza e comunque decisamente coperti da copyright. D'altra parte Google si difende dalle accuse e dichiara che è fisicamente impossibile tenere sotto controllo l'enorme mole di materiale che ogni giorno passa per i propri server. Tra le altre cose secondo Google appare chiaro dalle policy che l'utente rimane l'unico responsabile dei contenuti video che rende disponibili alla comunità. La polemica rimane dunque decisamente aperta.

RILASCIATO FAMILY.SHOW 2.0

Windows Presentation Foundation è una delle realtà più innovative dei nostri tempi, almeno per quanto riguarda l'ambiente Microsoft. La possibilità di programmare le applicazioni grafiche facendo ricorso all'uso del formato vettoriale invece del classico formato bitmap rappresenta una decisa evoluzione e un'occasione di innovazione per il programmatore. Per favorire il passaggio a WPF e spiegarne meglio tutte le funzionalità, era stato rilasciato il Family.Show 1.0. A farlo era stato Tim Sneath: client platform evangelist di Microsoft attraverso il suo blog http://blogs.msdn.com/tims/. Il Family.Show contiene una serie di esempi e di applicazioni attraverso il cui studio è semplice comprendere le modalità di programmazione relative a WPF. Nella nuova versione è stato aggiunto il supporto per i temi, le datagrid, il rich text e sono stati fissati alcuni bug presenti nella precedente versione

PYCON UNO, PER TUTTI I PYTHONISTI D'ITALIA

Il 9 e 10 giugno scorso si è tenuta a Firenze PyCon Uno, la prima conferenza italiana dedicata al linguaggio Python. La conferenza si era prefissa di divulgare Python e di dare visibilità agli sviluppatori professionisti, agli studenti, alle aziende che lo usano e a semplici curiosi. È stata quindi una conferenza aperta a tutti, non riservata a pochi specialisti. È stata organizzata da un gruppo di appassionati e senza finalità di lucro e ha avuto un successo inaspettato, circa 200 partecipanti venuti da tutta Italia. L'origine di tutto è stata molto informale: una e-mail mandata intorno a settembre dell'anno sorso a Valentino Volonghi, un ingegnere informatico milanese di 24 anni, con la quale alcuni utenti chiedevano se era possibile incontrarsi tutti insieme per parlare di programmazione Python. Dato che non era la prima volta che veniva fatta una tale richiesta, Valentino, che lavora da più di sette anni nel mondo Open Source ed è uno sviluppatore ufficiale di Twisted Matrix e di Nevow, si è preso l'incarico di organizzare una conferenza in piena regola. Lui e altri 20 programmatori conosciuti hanno preparato tutto in soli tre mesi. Gli sponsor, Google Inc., Sia SRL di Verona, Develer SRL, Link I.T. SPA, Python Software Foundation, Pragma 2000 SRL, Softwell SAS, MedMedia SRL e StatPro PLC, hanno contribuito alla realizzazione della conferenza sia finanziandola sia mettendo a disposizione il loro personale. Considerando che la conferenza ha avuto solo pochi mesi di preavviso e che 29 persone hanno proposto relazioni, non è difficile pensare a quanto stia diventando popolare Python in Italia. C'erano due keynote speaker. Uno era Alex Martelli di Google, che era anche uno degli organizzatori e si è fermato per tutta la conferenza, l'altro invece era Marco Pesenti Gritti, 27 anni, ex studente di Filosofia di Milano fondatore del progetto open source Galeon e attuale maintainer di Epiphany, un browser free basato su Mozilla. È arrivato solo per parlare ma ha portato anche due prototipi di OLPC. Il 23 Marzo 2007, a Firenze, è stata anche fondata da 20 sviluppatori come Lawrence Oluyede, Alex Martelli (Google), Carlo Miron (Visiant Galyleo), Marco Beri (Link I.T.) e altri la "Python Italia Associazione di Promozione Sociale", un'associazione che si prefigge di diventare un importante punto di riferimento per Python in Italia. "Abbiamo fondato l'associazione per avere una struttura più chiara nei confronti del mondo aziendale e produttivo e per ottenere rispetto dalle associazioni internazionali, come la Python Software Foundation è Google" dice Valentino, che ne è il presidente "perché è necessario essere trasparenti e davvero appassionati, se si vogliono realizzare delle cose." La nota carina fra tanti programmatori serissimi è che tutti aspettavano Marco Pesenti Gritti al banco di accettazione mentre lui era già seduto da mezz'ora nella sala dove avrebbe parlato, in prima fila. Erano già tutti pronti a fare un discorso di chiusura di riserva ma all'ultimo momento non è stato necessario. Il motivo? Gritti non si era registrato e, dato che nessuno lo conosceva, si era seduto ad aspettare che gli si desse la parola. Valentino è appena tornato da EuroPython, la conferenza europea di pythonisti che si è tenuta in Lituania. Gli abbiamo chiesto come vede il futuro di Python in Italia. "In Lituania erano presenti soltanto 250 sviluppatori. Già stiamo lavorando all'organizzazione di PyCon Due, che avverrà tra circa un anno. Speriamo di fare contenti ancora più sviluppatori e di promuovere nel migliore dei modi lo sviluppo di Python in Italia. Dopo la prima conferenza il nostro gruppo ha visto esplodere un gran numero di discussioni di alto livello, tanto da diventare un punto di riferimento non soltanto per la comunità Python ma anche per quelle di alcuni altri linguaggi come il C++. Con la prima conferenza abbiamo ottenuto successo inaspettato, però possiamo fare ancora di meglio e lo faremo."

Enrica Garzilli
http://Orientalia4All.net

GIÀ PRONTI PER WINDOWS 7

Non abbiamo ancora assorbito la metà delle novità introdotte in Windows Vista che già Microsoft ha annunciato i piani di sviluppo per il Windows che verrà. Non eravamo abituati a tanta celerità, ma a quanto pare l'intento dichiarato della società di Bill Gates è quello di accelerare sul piano dei rilasci e di fornire una roadmap più affidabile ai propri partner. La spinta verso questa decisione proviene soprattutto dal programma Software Assurance tramite il quale Microsoft consente ai suoi clienti di utilizzare sempre il software più recente suddividendo i pagamenti in rate annuali. Microsoft ha perciò dichiarato che Windows 7 sarà pronto per il 2010. Il nome in codice del progetto è attualmente "Blackcomb" ma già si preannuncia un cambiamento in "Vienna". Secondo quanto dichiarato il nuovo sistema operativo non sarà un semplice upgrade di Windows Vista ma piuttosto rappresenterà una radicale innovazione. Si pensa addirittura a fare sparire la storica barra introdotta già con Windows 95, sostituita da una per ora generica "nuova interfaccia". Dal punto di vista strettamente sistemistico le innovazioni potrebbero riguardare il file system, verrebbero invece mantenute le API 3D che hanno già fatto la loro comparsa in Windows Vista. Chi ha apprezzato i nuovi "ribbon" di Office 2007, per la verità non molti, saranno lieti di sapere che molto probabilmente lo stesso team che ha ridisegnato la nota suite per l'ufficio sarà anche incaricata di ridisegnare Internet Explorer. Si annota una significativa dichiarazione di Bill Gates il quale ha etichettato il prossimo venturo come "more user centric". Il boss di Microsoft spiega che attualmente chi si sposta da un PC all'altro ha necessità di reinstallare le applicazioni, sincronizzare i dati e compiere una serie di operazioni piuttosto noiose. Nella nuova visione l'idea è che esista una sorta di chiosco virtuale che diventi una specie di punto di raccolta per ciò che concerne le necessità dell'utente e garantisca una sorta di portabilità spinta anche quando si migra da un PC ad un altro. Infine qualcuna delle funzionalità che erano state progettate per Windows Vista e non hanno mai visto la luce saranno probabilmente spostate in Windows 7. Parliamo ad esempio della Sandbox che consente di utilizzare programmi non managed in una sorta di scatola virtuale isolata dal resto del sistema operativo e del riconoscimento e del completamento della scrittura, per essere chiari, qualcosa che assomigli a Google Suggest

20.000 DOLLARI PER IL NOKIA OPEN C CHALLENGE

L'annuncio è stato recentemente dato sul forum di Nokia e se ne può trovare la versione ufficiale all'indirizzo http://www.forum.nokia.com/main/resources/technologies/open_c/contest.html. Sono 20.000 i dollari messi in palio dal gigante della comunicazione in accordo con Symbian e Orange per colui il quale svilupperà la migliore applicazione per la piattaforma S60 utilizzando le librerie Open C. Si tratta di librerie nate in modo specifico per coloro che disponendo già di conoscenze in ambito C++ vogliano sviluppare per il mobile senza andare incontro a troppe difficoltà. Notevole anche lo sforzo di promuovere l'OpenSource, di fatto nelle regole del concorso compare una nota che rende obbligatorio l'uso di almeno un componente OpenSource all'interno del progetto sviluppato. Il vincitore del contest deve inoltre rendersi disponibile a partecipare allo SmartPhone Show che si terrà a Londra tra il 15 e il 17 Ottobre del 2007. Si tratta di un'iniziativa interessante che non solo promuove lo sviluppo di applicazioni innovative usando software OpenSource ma che lascia intravedere come l'agguerrita concorrenza derivante da Windows Mobile stia in una qualche misura costringendo i contendenti a ricercare soluzioni alternative che gli garantiscano di avere sempre uno spiraglio verso tecnologie che rendano più appetibili i propri prodotti per gli utenti finali.

GRANDE FRATELLO CON IL GPS

BASTA UN TELEFONINO E POCO ALTRO PER SAPERE SEMPRE DOVE SI TROVA CHI LO PORTA. SFRUTTA AL MASSIMO LE FUNZIONI DI WINDOWS MOBILE 5, AGGIUNGICI LE API DI GOOGLE EARTH, METTI TUTTO INSIEME ED IL GIOCO È FATTO!

Sicuramente ci è capitato di percorrere tratti autostradali dove abbiamo incrociato grossi autotreni su cui campeggiavano scritte tipo: "autoveicolo sottoposto a controllo satellitare" o comunque indicanti una sorta di tracciamento del veicolo via satellite. L'obiettivo del presente articolo è proprio quello di svelare una possibile infrastruttura che consenta di effettuare il tracciamento dei veicoli o di qualsiasi altro corpo in movimento in grado di trasportare un dispositivo mobile. La soluzione che andiamo ad implementare prevede diversi componenti: il "corpo" che andiamo a tracciare è in realtà un dispositivo mobile che monta il sistema operativo Windows Mobile 5 o Windows Mobile 6 ed è dotato di un'antenna per la ricezione delle informazioni satellitari. L'antenna non deve essere necessariamente integrata nel dispositivo quindi possiamo, ad esempio, tranquillamente utilizzare le antenne che usano la tecnologia Bluetooth per connettersi ai diversi dispositivi. In questo caso, ovviamente, anche il dispositivo che andiamo ad utilizzare deve supportare la tecnologia Bluetooth. Il dispositivo mobile deve altresì essere in grado di connettersi ad un web service verso cui trasmette la sua posizione in tempo reale. Tipicamente, questa connessione è possibile nel caso in cui il dispositivo abbia anche un supporto telefonico; in questo modo è possibile connettersi alle reti di trasmissione dati che si appoggiano alle tecnologie GPRS piuttosto che UMTS. Le tracce ricevute sono riprodotte in tempo reale su di una pagina di controllo che mostra i dispositivi su di una mappa; la georeferenziazione dei dispositivi è possibile grazie ai servizi disponibili in rete che, nella fattispecie, sono Google Maps e Microsoft Virtual Earth.

La complessità della soluzione ci consente di analizzare diversi concetti e tecnologie. Avremo modo di sviluppare un web service (per la ricezione delle posizioni), una applicazione mobile che si interfaccia con apparati GPS, una pagina web di controllo che utilizza AJAX per aggiornare in tempo reale ed in modo "fluido" le tracce dei nostri dispositivi.

Proprio a causa della complessità della soluzione (e del relativo spazio necessario per la sua analisi), l'articolo verrà suddiviso in due parti. In questa prima parte andremo ad analizzare l'applicazione mobile ed il web service che riceve le posizioni dei dispositivi.

Fig. 1: Schema della soluzione

L'OBIETTIVO

Nella **Figura 1** è raffigurato uno schema di massima della soluzione che intendiamo realizzare. Sulla parte sinistra sono rappresentati i dispositivi che, tramite l'interazione con l'antenna satellitare ed il relativo supporto per la rete GPS (Global Positioning System), sono in grado di rilevare in tempo reale la loro posizione in termini di latitudine e longitudine. La posizione viene inviata ad un web service che, a sua volta, utilizza una classe "wrapper" per memorizzare il dato in modo persistente. La *classe* "wrapper" non è nient'altro che una *classe* che incapsula la logica di persistenza del dato verso una qualsivoglia base dati e la sua creazione è frutto dell'architettura "a strati" (n-tier) su cui si basa la soluzione che vogliamo sviluppare. La classe espone i metodi per interagire con la base dati e, di conseguenza, rende il resto

della soluzione indipendente dalla base dati prescelta. Nella soluzione che stiamo analizzando, la base dati sarà ospitata su SQL Server 2005; laddove volessimo utilizzare un qualsiasi altro prodotto (MySql piuttosto che Oracle), ma anche se volessimo memorizzare le posizioni su di un disco rigido (in un file XML piuttosto che in un file testuale), non dovremmo far altro che implementare la classe wrapper in modo che questa esponga i metodi per interagire con la base dati di nostra scelta.
Per consentire il controllo delle tracce, utilizziamo una pagina web che interroga la classe wrapper per ricavare le posizioni di tutti i dispostivi, mentre utilizza i servizi messi a disposizione da Google Maps o Microsoft Virtual Earth per georeferenziare i dispositivi su di una mappa.

LA BASE DATI E LA CLASSE WRAPPER

Nella implementazione corrente, come base dati scegliamo SQL Server 2005, quindi dobbiamo creare un nuovo database che chiamiamo Tracker (ai soli fini dell'esecuzione dell'esempio possiamo lasciare le impostazioni predefinite relative alla dimensione, il log, la posizione e gli altri parametri del database; ovviamente una volta che decidiamo di porre in produzione la soluzione, dovremo pianificare opportunamente la creazione e la configurazione dei suddetti parametri).
Lo schema della base dati è molto semplice ed è composto da due tabelle: **Devices** che contiene l'elenco dei dispositivi definiti tramite un identificativo ed una descrizione; **Positions** che contiene le posizioni in termini di latitudine e longitudine di ogni dispositivo abbinato tramite il relativo identificativo. Poniamo la tabella delle posizioni in relazione con la tabella dei dispositivi attraverso il relativo identificativo (si faccia riferimento alla **Figura 2**). Per implementare la classe wrapper, creiamo un nuovo progetto di tipo "**Class Library**" e lo nominiamo *TrackerObject* (il codice della classe è contenuto nel file *TrackerObject.cs* reperibile nei file a supporto dell'articolo). Dato che utilizzeremo la classe per interagire con la base dati definita precedentemente, definiamo tre membri privati che rappresentano rispettivamente un oggetto di tipo *SqlConnection* (*_connection*), usato per stabilire una connessione verso la base dati, un oggetto di tipo *SqlCommand* (*_command*), usato per inviare i comandi alla base dati ed un oggetti di tipo *StringBuilder*

Positions

Column Name	Condensed Type	Nullable
idDevice	int	No
latitude	varchar(50)	No
longitude	varchar(50)	No

FK_Positions_Devices

Devices

Column Name	Condensed Type	Nullable
idDevice	int	No
Description	varchar(50)	No

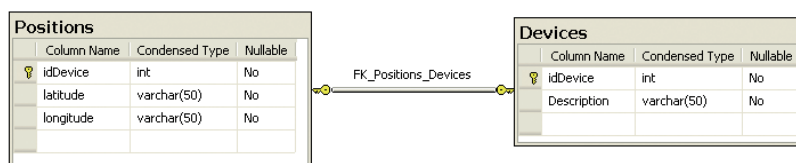



***Fig. 2:** Lo schema della base dati che contiene le posizioni dei dispositivi.*

(*_query*), usato per compilare il testo del comando da inviare alla base dati.

```
namespace TrackerObject
{
 public class TrackerObject
 {
  SqlConnection _connection;
  SqlCommand _command;
  StringBuilder _query;
. . . .
 }
}
```

Nel costruttore della classe definiamo la stringa di connessione alla base dati e creiamo l'istanza dell'oggetto di tipo *SqlConnection* che useremo nei metodi della classe. Per semplicità si è scelto di definire in modo statico la stringa di connessione, ovviamente possiamo rendere più flessibile la soluzione sia passando un parametro al costruttore sia ricavando la stringa di connessione da un file di configurazione.

```
public TrackerObject()
{
 // Definiamo la stringa di connessione alla base dati
 string strConnection = "data source=localhost;initial
                                  catalog=Tracker;User
 ID=TrackerUsr;Password=TrackerUsr";
 // Definiamo una connessione verso la base dati
 _connection = new SqlConnection(strConnection);
}
```

La classe *TrackerObject* possiede due metodi; utilizziamo il metodo *SavePosition* per salvare nella base dati la posizione di un dispositivo espressa tramite i valori di longitudine e latitudine. Il metodo riceve in input l'identificativo del dispositivo (*idDevice*) ed i valori di latitudine (*latitude*) e longitudine (*longitude*) espressi nel formato decimale.
Nella costruzione del testo del comando da inviare alla base dati, dobbiamo prevedere di cancellare preventivamente dalla tabella delle posizioni, l'eventuale posizione memorizzata precedentemente:

```
DELETE FROM Positions WHERE idDevice = xxx
```

Di seguito compiliamo il commando di inserimento, utilizzando i parametri di input, quindi creiamo

SQL SERVER 2005 EXPRESS EDITION

Per provare la soluzione che svilupperemo nella presente trattazione, è necessario installare il prodotto Sql Server 2005 che, nella sua versione gratuita (utilizzabile solo per uso personale), è scaricabile a partire dall'indirizzo: http://www.microsoft.com/downloads/details.aspx?familyid=220549b5-0b07-4448-8848-dcc397514b41&displaylang=en.

BASE DATI TRACKER

Al fine di facilitare il lettore nella creazione della soluzione, nel materiale a supporto dell'articolo è presente il file Creazione DB Tracker.sql che contiene gli script in linguaggio Transact-SQL che consentono la creazione della base dati e delle relative tabelle.

l'istanza dell'oggetto di tipo *SqlCommand* passando al suo costruttore il testo del comando appena prodotto e l'istanza dell'oggetto di tipo *SqlConnection* creata in fase di creazione della classe. Per inviare il comando alla base dati utilizziamo il metodo *ExecuteNonQuery* dell'oggetto di tipo *SqlCommand* opportunamente inserito tra i comandi di apertura e successiva chiusura della connessione verso la base dati. Anche in questo caso, per semplicità, abbiamo evitato di gestire gli eventuali errori causati dall'invio del comando; in ambiente di produzione è consigliabile inserire l'esecuzione del comando in un blocco *Try-Catch* e gestire l'eventuale errore da inviare al chiamante.

```
public void SavePosition(int idDevice, double latitude,
                                        double longitude)
{
 // Costriuamo il testo del comando da
 // inviare alla base dati per effettuare il salvataggio
 // della posizione del  dispositivo
 _query = new StringBuilder("DELETE FROM Positions
                                  WHERE idDevice = ");
 _query.Append(idDevice).Append("; ");
 _query.Append("INSERT INTO Positions (idDevice,
                                  latitude, longitude) ")
        .Append(" VALUES ")
        .Append("(")
        .Append(idDevice)
        .Append(", '")
        .Append(latitude)
        .Append("', '")
        .Append(longitude)
        .Append("'); ");
 // Costruiamo il comando
 _command = new SqlCommand(_query.ToString(),
                                        _connection);
 _connection.Open();
 // Eseguiamo il comando
 _command.ExecuteNonQuery();
 _connection.Close();
}
```

Per ricavare la lista delle posizioni dei dispositivi che vogliamo controllare, invochiamo il secondo metodo della classe *TrackerObject*, ovvero *GetPositions*. Il metodo non ha parametri di input dato che, come risultato, restituisce tutte le posizioni memorizzate nella base dati. Il valore di ritorno del metodo è una stringa; questa scelta è giustificata dal fatto che, come vedremo nel seguito, il risultato dell'invocazione, verrà utilizzato nell'ambito di una applicazione web basata su AJAX quindi in funzioni sviluppate in linguaggio Javascript. In questa situazione è opportuno affidarsi a strutture dati il più semplice possibile in modo da poter gestire le informazioni anche in ambiti diversi. Se vogliamo, possiamo pensare che il valore di ritorno del metodo è "serializzato" dal metodo stesso. Come detto dobbiamo ritornare una lista di posizioni che sono caratterizzate da una latitudine, una longitudine e, naturalmente, dal riferimento al dispositivo a cui si riferiscono. Possiamo allora pensare di rappresentare una generica posizione con una stringa formata dalla giustapposizione dei seguenti valori: identificativo del dispositivo, descrizione del dispositivo, latitudine, longitudine. Per individuare le singole componenti dell'informazione possiamo suddividere i valori per mezzo del segno "^" (accento circonflesso); di conseguenza una singola posizione può essere rappresentata, per esempio, con la seguente stringa:

```
1^Primo Dispositivo^ 43,61783^ 13,52374
```

La prima parte del metodo *GetPositions*, si occupa di compilare il testo del comando da inviare alla base dati, in modo che il recordset di dati ritornato, sia composto in effetti da elementi che seguono il formato precedente.

```
public string GetPositions()
{
 // Costriuamo il testo del comando da inviare
 // alla  base dati per effettuare la richiesta
 // di tutte le posizioni
 _query = new StringBuilder(
    "SELECT CAST(d.idDevice AS varchar(10)) + '^'
                          + d.Description + '^' + ");
 _query.Append("CAST(p.latitude AS varchar(20)) +
                                            '^' + ")
        .Append("CAST(p.longitude AS varchar(20))
                                  AS TrackerData ")
        .Append("FROM Positions p ")
        .Append("INNER JOIN Devices d ON d.idDevice
                                   = p.idDevice ");
 // Costruiamo il comando
 _command = new SqlCommand(_query.ToString(),
                                        _connection);
 _connection.Open();
 // Eseguiamo il comando che restituisce
 // un oggetto SqlDataReader
 SqlDataReader sdr = _command.ExecuteReader();
 // Oggetto che conterrà la stringa delle posizioni
 StringBuilder sbPositions = new StringBuilder();
 // Scorriamo il datareader
 while (sdr.Read())
 {
    // Estraiamo il dato
    string sData = sdr["TrackerData"].ToString();
    sData = sData.Replace(',', '.');
    // Accodiamo il dato
    sbPositions.Append(sData).Append("|");
 }
 sdr.Close(); _connection.Close();
 return sbPositions.ToString();
}
```

Di seguito, creiamo l'istanza dell'oggetto di tipo *SqlCommand* passando al suo costruttore il testo del comando appena prodotto e l'istanza dell'oggetto di tipo *SqlConnection* creata in fase di creazione della classe. Per inviare il comando alla base dati utilizziamo il metodo *ExecuteReader* che ritorna un oggetto di tipo *SqlDataReader* contenente i record delle posizioni dei nostri dispositivi. A questo punto non ci resta che scorrere l' oggetto di tipo *SqlDataReader* per produrre la stringa finale da tornare al chiamante. Ogni singola posizione verrà giustapposta e separata dal carattere "|" ("pipe"); in questo modo la stringa finale delle posizioni potrà essere, per esempio, la seguente:

```
1^Primo Dispositivo^43,61783^13,52374|
2^Secondo Dispositivo^44,43323^13,44238| . . .
```

Ricapitolando, se vogliamo implementare la classe *TrackerObject* dobbiamo produrre il codice relativo ai due metodi *SavePosition* e *GetPositions*; questa indicazione ci può tornare utile qualora volessimo cambiare la base dati in cui persistere le posizioni dei nostri dispositivi. Basterà infatti sviluppare un nuovo progetto di tipo "Class Libray" ed avere l'accortezza di nominarlo sempre *TrackerObject*; dopodiché non dovremo far altro che implementare i due metodi in modo che le informazioni da salvare e le relative interrogazioni vengano dirette al nuovo supporto che, per esempio, potrebbe essere anche un file XML salvato nel disco rigido del server che ospita la soluzione.
Dopo aver compilato il progetto non dobbiamo far altro che sostituire la libreria (*TrackerObject.dll*) che abbiamo analizzato con la nuova da noi prodotta; da questo momento i dati verranno salvati nella nuova base dati.

IL WEB SERVICE TRACKERWEBSERVICE

Per implementare il web service, da Visual Studio 2005 creiamo un nuovo progetto di tipo "ASP.NET Web Service Application" (il codice del web service è reperibile nel file *TrackerWS.asmx.cs*). Il sistema crea tutta una serie di file e propone in automatico il primo semplice metodo da esporre. Per i nostri scopi, il web service deve esporre un metodo che consenta di salvare la posizione di un dispositivo. Al fine di rendere questo componente indipendente dalla base dati in cui verranno memorizzate le posizioni, utilizziamo la classe wrapper appena creata quindi, dobbiamo innanzitutto aggiungere il riferimento alla libreria che contiene il codice della classe (nel pannello "Solution Explorer" facciamo clic con il tasto destro del mouse sul nome del progetto, quindi selezioniamo "Add Reference …") e di seguito dobbiamo aggiungere una clausola using in testa al codice.

```
using System;
using System.Data;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.ComponentModel;
using TrackerObject;
namespace TrackerWebService
{
 /// <summary>
 /// Summary description for Service1
 /// </summary>
 [WebService(Namespace = "http://tempuri.org/")]
 [WebServiceBinding(ConformsTo =
                              WsiProfiles.BasicProfile1_1)]
 [ToolboxItem(false)]
 public class TrackerWS :
                         System.Web.Services.WebService
 {
  [WebMethod]
  public void SavePosition(int idDevice, double
                              latitude, double longitute)
  {
   // Creiamo una istanza dell'oggetto wrapper
   TrackerObject.TrackerObject oTacker = new
                         TrackerObject.TrackerObject();
   // Salviamo la posizione nella base dati
   oTacker.SavePosition(idDevice, latitude, longitute);
  } }}
```

PIÙ VELOCI CON STRINGBUILDER

Se analizziamo attentamente i vari frammenti di codice possiamo notare che, in corrispondenza delle operazioni di concatenamento di stringhe, in luogo del "semplice" operatore "+" viene utilizzato un oggetto di tipo StringBuilder ed il relativo metodo Append. Questa scelta non è un caso dato che il suddetto oggetto ha una resa in termini di prestazioni estremamente più elevata del corrispondente operatore "+". L'operatore "+", ogni qualvolta effettua un concatenamento, alloca in memoria un spazio pari all'effettiva somma dei componenti che concateniamo, quindi copia fisicamente i componenti producendo la stringa concatenata; ciò implica che in una concatenazione di 1000 elementi, il primo elemento viene copiato 1000 volte, il secondo 999 e così via. L'oggetto di tipo StringBuilder invece effettua una concatenazione "reale" dei diversi componenti della stringa, ed ogni componente non viene mai copiato per produrre la nuova concatenazione. È facilmente intuibile come, in una situazione in cui si debbano fare diverse concatenazioni (come nel caso in esame), la resa dell'oggetto di tipo StringBuilder sia assolutamente da preferire.

L'unico metodo esposto dal web service è *SavePosition* che riceve in input l'identificativo del dispositivo, una latitudine ed una longitudine; queste due ultime grandezze espresse nel formato decimale. Il corpo del metodo è banale dato che dobbiamo semplicemente creare una nuova istanza della classe *TrackerObject*, quindi dobbiamo invocare il suo metodo *SavePosition* a cui passiamo i parametri che riceviamo in input.

L'APPLICAZIONE MOBILE

L'applicazione deve interagire con la rete satellitare GPS (Global Positioning System) e comunicare la posizione rilevata tramite l'invocazione di un web service. Questa semplice affermazione ci fa capire che i dispositivi su cui andremo a distribuire la nostra applicazione dovranno essere dotati di antenna di ricezione satellitare (sia integrata che connessa tramite tecnologia Bluetooth) e di supporto telefonico per comunicare i dati via GPRS/UMTS.
Per quanto riguarda l'interazione verso l'hardware GPS, con l'uscita del sistema operativo Windows Mobile 5, Microsoft ha introdotto nel sistema, il "*GPS Intermediate Driver*" che non è nient'altro che uno strato di software che astrae l'apparato GPS dal dispositivo. Ciò ci consente (in teoria) di sviluppare applicazioni che potranno interagire con qualsiasi tipo di apparato GPS.
Posta in questi termini, la questione sembra molto interessante, purtroppo chi è ormai abituato a lavorare con il framework .Net ed il C# (come l'autore dell'articolo) avrà una brutta sorpresa, infatti la nutrita quantità di funzioni disponibili per interagire con gli apparati GPS, è rivolta agli sviluppatori "nativi" ovvero a coloro che sviluppano in C++.
Fortunatamente è prassi consolidata della Microsoft di fornire insieme ai suoi prodotti, tutta una serie di supporti per gli sviluppatori, che normalmente vanno sotto il nome di Software Developer's Kit (SDK). Anche in occasione dell'uscita del sistema operativo Windows Mobile 5, Microsoft ha rilasciato il relativo SDK (si veda la nota a fianco pagina) che contiene, tra i vari esempi, anche una soluzione che consente di invocare le funzionalità native relative al GPS Intermediate Driver da una applicazione C#.
La soluzione è composta da una libreria di classi (Microsoft.WindowsMobile.Samples.Location) e da una applicazione mobile da utilizzare per testare le diverse funzionalità messe a disposizione dalla libreria.

ANCHE NELLE MIGLIORI FAMIGLIE ...

Sebbene la provenienza dell'SDK sia una garanzia più che notevole, in realtà, proprio nella libreria che dobbiamo utilizzare, è presente un errore di programmazione a cui dobbiamo porre rimedio.
Apriamo il progetto *Microsoft.WindowsMobile.Samples.Location*, quindi il file *DegreesMinutesSeconds.cs* che contiene la omonima classe. Se ci portiamo sul metodo *ToDecimalDegrees* possiamo osservare il seguente codice:

```
public double ToDecimalDegrees()
{
 int absDegrees = Math.Abs(degrees);
 double val = (double)absDegrees +
     ((double)minutes / 60.0) +
     ((double)seconds / 3600.0);
 return val * (absDegrees / degrees);
}
```

Il metodo è molto semplice ed è utilizzato per convertire le coordinate geografiche dal formato Gradi/Minuti/Secondi (DMS) al formato decimale.
Senza addentrarci nella regola di conversione, possiamo notare abbastanza facilmente che il metodo produrrà sicuramente un errore nel caso in cui la posizione rilevata sia nei pressi di Greenwich ovvero in una posizione a zero Gradi. Nell'ultima riga di codice, infatti si ottiene una eccezione *DivideByZeroException* a causa della divisione per zero. Esistono diverse discussioni aperte nei forum che forniscono una soluzione abbastanza semplice del problema. In realtà però il problema è ben altro.
Il metodo in questione viene fortemente utilizzato nella nostra soluzione dato che i servizi web che andremo ad usare per tracciare la nostra posizione, accettano proprio le coordinate in formato decimale. Ebbene se utilizziamo il metodo così com'è, il posizionamento viene errato clamorosamente. Da una analisi dei risultati ottenuti per il posizionamento nel nostro paese (nella zona di Ancona per essere precisi) è emersa una errata valorizzazione delle variabili *degrees*, *minutes* e *seconds*; in particolare la variabile *degrees* contiene il valore 0, la variabile *minutes* contiene il realtà il valore dei Gradi, la variabile *seconds* contiene in realtà il valore dei Minuti.

GPS INTERMEDIATE DRIVER

Una trattazione esaustiva relativa all'uso del "GPS Intermediate Driver" è reperibile a partire dall'indirizzo http://msdn2.microsoft.com/en-us/library/bb202086.aspx

SOLUZIONE GPS IN WINDOWS MOBILE 5.0 SDK

Il pacchetto "Windows Mobile 5.0 SDK for Pocket PC" è scaricabile gratuitamente dal sito Microsoft a partire dall'indirizzo: http://www.microsoft.com/downloads/details.aspx?familyid=83A52AF2-F524-4EC5-9155-717CBE5D25ED&displaylang=en**.**

Supponendo di aver installato il pacchetto nella cartella predefinita, è possibile reperire l'esempio relativo alla gestione del GPS a partire dalla seguente cartella: C:\Programmi\Windows CE Tools\wce500\Windows Mobile 5.0 Pocket PC SDK\Samples\Cs\Gps.

Al fine di avere i giusti valori del posizionamento, è necessario allora modificare il metodo come segue:

```
public double ToDecimalDegrees()
{
 double retVal;
 retVal = minutes + seconds / 60;
 return retVal;
}
```

Con la precedente implementazione i risultati tornano ad essere esatti.

I RIFERIMENTI ALLE RISORSE

Tornando all'applicazione, dato che l'interfaccia che andiamo a sviluppare non necessita di particolari controlli grafici, possiamo tranquillamente utilizzare il template *Device Application (1.0)* che, sebbene si basi sulla versione 1.0 del Compact Framework .Net, ci consente di NON dover installare ulteriori componenti sul dispositivo che, ricordiamolo, deve montare il sistema operativo Windows Mobile 5 o Windows Mobile 6. Nella soluzione fornita a supporto del presente articolo il progetto che implementa l'applicazione mobile si chiama *TrackerMobile*.
Come detto, l'applicazione deve interfacciarsi con un apparato di navigazione satellitare GPS e con un web service; per tale ragione, come prima cosa, andiamo a definire i riferimenti alle relative risorse. Per l'interfacciamento con l'apparato GPS, dobbiamo aggiungere un riferimento alla libreria *Microsoft.WindowsMobile.Samples. Location*; nel pannello "Solution Explorer", facciamo

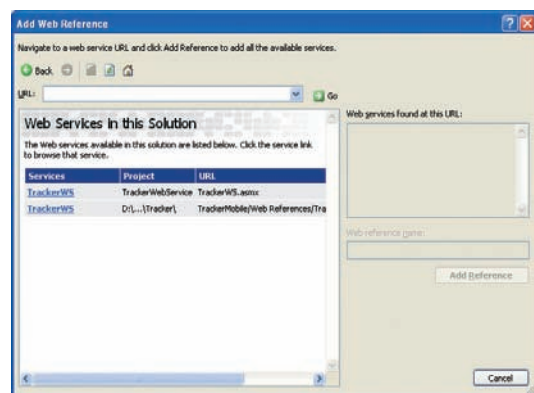



Fig. 3: Selezioniamo il servizio TrackerWebService.

clic con il tasto destro del mouse sul nome del progetto (*TrackerMobile*) quindi facciamo clic sulla voce "*Add Reference …*"; dal dialogo che ci appare possiamo selezionare la libreria navigando nella struttura delle cartelle. Per quanto riguarda il web service, dobbiamo aggiungere un riferimento al servizio *TrackerWebService* che abbiamo sviluppato in precedenza; sempre nel pannello "Solution Explorer", facciamo clic con il tasto destro del mouse sul nome del progetto (*TrackerMobile*) quindi facciamo clic sulla voce "Add Web Reference …"; dal dialogo che ci appare possiamo scegliere di cercare il servizio o nella soluzione corrente, o nella macchina locale, o nella rete locale. Per comodità possiamo scegliere di cercare nella soluzione corrente in modo da avere come unica scelta il servizio *TrackerWebService*. Se selezioniamo il servizio, Visual Studio ci mostra la pagina di test dove sono elencati i metodi esposti dal servizio. Dallo stesso dialogo possiamo dare un nome significativo al riferimento (TrackerWS) tramite il campo "Web reference name" ed aggiungere il riferimento premendo il bottone "Add Reference" (Figura 4).

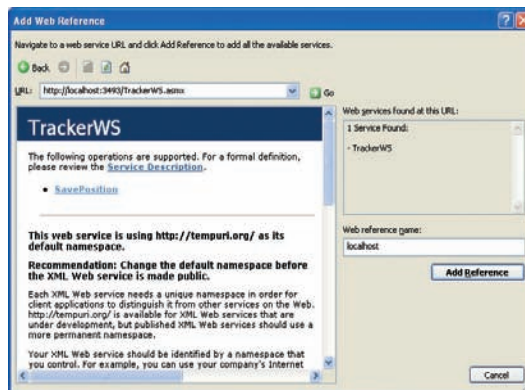



Fig. 4: Aggiungiamo il riferimento al servizio TrackerWebService

Se abbiamo effettuato correttamente tutte le operazioni, il nostro progetto deve contenere i riferimenti mostrati in Figura 5.

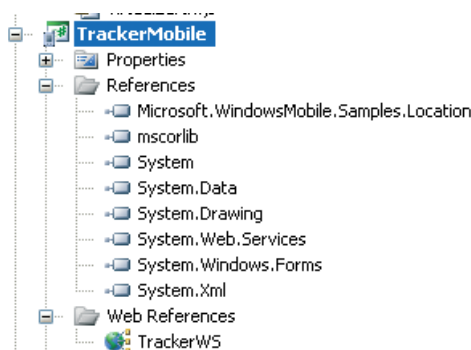



Fig. 5: I riferimenti dell'applicazione TrackerMobile.

FORMATO DECIMALE DELLE COORDINATE GEOGRAFICHE

Al fine di produrre un valore più "gestibile" dai programmi di elaborazione dati, si preferisce spesso esprimere le coordinate geografiche non nel formato Gradi/Minuti/Secondi (DMS) ma nel corrispettivo formato decimale. La formula di conversione è la seguente:

```
Valore Decimale = Gradi +
  (Minuti*1/60) + (Secondi*1/3600)
```

L'INTERFACCIA UTENTE ED IL CODICE

Iniziamo ad analizzare l'applicazione che dobbiamo sviluppare partendo dall'interfaccia grafica rappresentata in Figura 6.
Sono individuabili tre pannelli utilizzati per raggruppare in modo coerente le funzioni e le informazioni da mostrare all'utente.

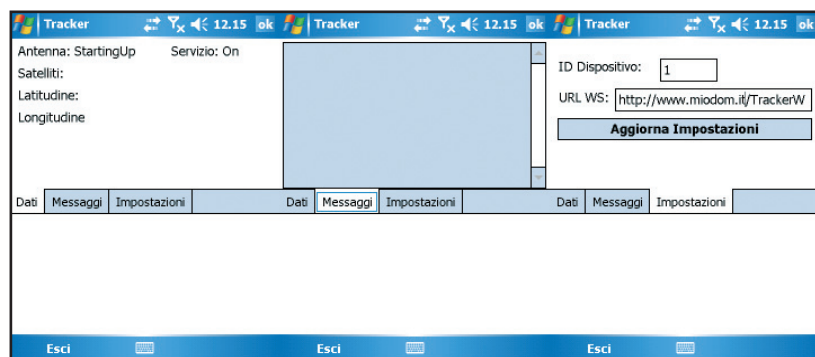



Fig. 6: L'applicazione TrackerMobile.

Nel primo pannello (*Dati*) vengono mostrate le informazioni ricavate dall'apparato satellitare. Oltre allo stato dell'apparato, si mostrano il numero dei satelliti agganciati e la posizione geografica in termini di latitudine e longitudine espresse nel formato decimale.
Il secondo pannello (*Messaggi*) è riservato ai messaggi di sistema che vengono mostrati all'utente in caso di errori. Nell'ultimo pannello (*Impostazioni*) possiamo configurare l'applicazione indicando l'identificativo del dispositivo e l'indirizzo del web service a cui inviare la nostra posizione (vedremo che queste impostazioni sono memorizzate in un file XML che viene caricato in fase di avvio dell'applicazione). Per analizzare il codice facciamo riferimento al file *Tracker.cs*. Nella dichiarazione della form Tracker, dobbiamo definire una serie di variabili globali che utilizzeremo in fase di esecuzione: dichiariamo un gestore degli eventi per gestire l'aggiornamento dei dati relativi alla nostra posizione; una serie di variabili che conterranno le informazioni ricevute dall'apparato GPS; l'istanza del web service *TrackerWS*; alcune variabili di configurazione.

```
public class Tracker : System.Windows.Forms.Form
{
//Gestore Eventi per l'aggiornamento dei dati
private EventHandler updateDataHandler;
// Variabili GPS
GpsDeviceState device = null;
GpsPosition position = null;
Gps gps = new Gps();
// Web Service
TrackerWS.TrackerWS oTrackerWS;
// Variabili di configurazione
string fullyQualifiedName = "";
XmlDocument oXConfigDoc = new XmlDocument();
string TrackerDevice = "0";
string TrackerUrlWS = "";
. . .
}
```

Nel costruttore della form, la prima riga di codice (inserita automaticamente all'atto della creazione del progetto) è relativa alla costruzione degli elementi dell'interfaccia grafica ed il relativo metodo (*InitializeComponent*) è compilato automaticamente da Visual Studio 2005 quando andiamo ad interagire con i controlli nella pagina di design della form.
Di seguito dobbiamo attivare la configurazione dell'ambiente per mezzo dei due metodi *InitializeConfiguration* e *InitializeWS* (analizzeremo il relativo codice in seguito). Nel caso in cui la configurazione vada a buon fine, possiamo attivare il meccanismo di tracciamento associando all'istanza gps due gestori degli eventi che terranno sotto controllo le variazioni dello stato dell'apparato e le variazioni della posizione. Per correttezza è giusto precisare che questa ultima parte del codice è ricavata dall'esempio del Software Developer's Kit (SDK) già citato in precedenza.

```
public Tracker()
{
 InitializeComponent();
 if (InitializeConfiguration() &&
  InitializeWS())
 {
  // Aggiorniamo i controlli relativi
  // alla impostazioni di sistema
  tbIdDevice.Text = TrackerDevice;
  tbUrlWS.Text = TrackerUrlWS;
  // Creiamo l'istanza del gestore
  // dell'evento di aggiornamento dei dati
  updateDataHandler = new
                              EventHandler(UpdateData);
  // Definiamo i gestori degli eventi
 // per intercettare la modifica
  // dello stato dell'apparato GPS e la modifica della
                                      posizione rilevata
  gps.DeviceStateChanged += new
                         DeviceStateChangedEventHandler
                                (gps_DeviceStateChanged);
  gps.LocationChanged +=     new
  LocationChangedEventHandler(gps_LocationChanged);
  if (!gps.Opened)
  {
   gps.Open();
  }
 }
}
```

LA CONFIGURAZIONE DELL'APPLICAZIONE

I file di configurazione sono da sempre il veicolo per adattare una applicazione allo specifico ambiente in cui opera. Se una volta era prassi utilizzare i file ".ini", negli ultimi anni è divenuto naturale esprimete i parametri di configurazione tramite file XML. Anche per la nostra applicazione utilizzeremo il file *TrackerMobile.xml* per esprimere i parametri necessari ad ogni dispositivo. In particolare definiamo due parametri ovvero l'identificativo del dispositivo che passeremo al web service insieme alla posizione rilevata, e l'indirizzo di pubblicazione del web service.

```
<TrackerMobile>
 <TrackerDevice id="1" />
 <TrackerWS
                    url="http://www.mioddominio.it/
                         TrackerWS/TrackerWS.asmx" />
</TrackerMobile>
```

Usiamo il metodo *InitializeConfiguration* per caricare il file XML e valorizzare le relative variabili che verranno usate nel resto del codice.

```
private bool InitializeConfiguration()
{
 bool retCode = true;
 // Individuiamo il percorso del
 // file di configurazione
 fullyQualifiedName =
 (Assembly.GetExecutingAssembly().GetModules())[0]
   .FullyQualifiedName;
 // Sostituiamo il suffisso "exe" con il suffisso "xml"
 string configPath =
             fullyQualifiedName.Replace(".exe", ".xml");
 try
 {
  // Carica il documento di configurazione
  oXConfigDoc.Load(configPath);
  // Selezioniamo il nodo contenente l'identificativo
                                        del dispositivo
  XmlNode oXmlNodeTrackerDevice = oXConfigDoc.
    GetElementsByTagName("TrackerDevice")[0];
  // Memorizziamo l'identificativo del dispositivo
  TrackerDevice =
        oXmlNodeTrackerDevice.Attributes["id"].Value;
 }
 catch(Exception exc)
 {
    lblMess.Text += " Errore nel caricamento del file
                   di configurazione: " + exc.Message;
    retCode = false;
 }
 return retCode;
}
```

Nel metodo *InitializeConfiguration*, il problema più difficile da risolvere è il reperimento del file di configurazione da cui trarre i parametri di configurazione. Se imponiamo che il file deve essere memorizzato nella stessa cartella dell'eseguibile e che debba avere lo stesso nome dell'eseguibile, possiamo facilmente risolvere il problema utilizzando la seguente linea di codice:

```
fullyQualifiedName =
 (Assembly.GetExecutingAssembly().GetModules())[0]
   .FullyQualifiedName;
```

La variabile *fullyQualifiedName* conterrà il percorso completo dell'eseguibile; la successiva sostituzione del suffisso "exe" con il suffisso "xml" ci fornirà il percorso al file di configurazione. Nel resto del metodo non dobbiamo far altro che caricare il contenuto del file di configurazione nella variabile globale *oXConfigDoc* e valorizzare la variabile globale *TrackerDevice* con l'identificativo che sarà assegnato al dispositivo. Con il metodo *InitializeWS* creiamo e configuriamo l'istanza dell'oggetto che utilizzeremo per invocare il web service a cui passeremo la posizione del dispositivo.

```
private bool InitializeWS()
{
 bool retCode = true;
 // Creiamo una istanza del web service
 oTrackerWS = new
              TrackerMobile.TrackerWS.TrackerWS();
 // Impostiamo il proxy
 WebProxy objProxy = new WebProxy();
 oTrackerWS.Proxy = objProxy;
 GlobalProxySelection.Select = objProxy;
 try
 {
 // Selezioniamo il nodo del documento
 // di configurazione che contiene le informazioni
 // relative al web service
  XmlNode oXmlNodeTrackerWS =
oXConfigDoc.GetElementsByTagName("
                         TrackerWS")[0];
  // Impostiamo l'indirizzo del web service
  oTrackerWS.Url =
          oXmlNodeTrackerWS.Attributes["url"].Value;
  // Memorizziamo l'indirizzo del web service
  TrackerUrlWS =
          oXmlNodeTrackerWS.Attributes["url"].Value;
 }
 catch
 {
    lblMess.Text += "Errore nella inizializzazione ";
    retCode = false;
 }
 return retCode;
}
```

Da notare che, oltre a creare una istanza dell'oggetto di tipo *TrackerWS*, dobbiamo anche definire un proxy (anche se non lo usiamo effettivamente) in modo che l'applicazione possa correttamente colloquiare con la risorsa web. Attraverso la proprietà *Url* dell'oggetto di tipo *TrackerWS* imponiamo, infine, il corretto indirizzo di pubblicazione del web service; questa operazione ci consente di gestire in modo dinamico e parametrico l'accesso alla risorsa.

L'INVIO DELLA POSIZIONE CORRENTE

La parte rilevante della nostra applicazione è sicuramente rappresentata dal codice che gestisce il rilevamento della posizione ed il relativo invio al web service. Dato che anche questa parte di codice è stata rilevata dall'esempio del Software Developrs Kit di Microsoft, ci limiteremo ad analizzare le parti modificate in modo da poter funzionare per la nostra applicazione.
Nel costruttore della classe *Tracker*, abbiamo definito due gestori degli eventi per intercettare la variazione dello stato dell'apparato GPS e la variazione della posizione rilevata.
I metodi *gps_LocationChanged* e *gps_DeviceStateChanged* invocano un ulteriore gestore, *UpdateData*, attraverso il metodo *Invoke*; in questo modo possiamo aggiornare l'interfaccia utente nello stesso thread. Nel caso della variazione della posizione, all'interno del gestore *gps_LocationChanged* viene valorizzata una variabile di tipo *GpsPosition* che verrà poi elaborata dal metodo *UpdateData*.

```
protected void gps_LocationChanged(object
            sender, LocationChangedEventArgs args)
{
 position = args.Position;
 // Chiama il metodo UpdateData
 // attraverso updateDataHandler
 // in modo da poter aggiornare
 // l'interfaccia utente nello stesso thread
 Invoke(updateDataHandler);}
```

Attraverso il metodo *UpdateData* possiamo aggiornare sia i controlli presenti nel pannello *Dati* sia inviare la posizione al web service. Se andiamo ad analizzare il frammento di codice relativo alla gestione della posizione, attraverso le numerose proprietà della variabile position (di tipo *GpsPosition*) possiamo controllare la validità dei dati ricevuti per mostrarli all'utente andando a valorizzare i controlli dell'interfaccia. Per inviare la posizione al web service invochiamo il metodo SavePosition dell'oggetto *oTrackerWS* a cui passiamo l'identificativo del dispositivo e le coordinate geografiche rappresentate in termini di latitudine e longitudine espresse nella notazione decimale.

```
// Si controlla la validità del dato rilevato
if (position.LatitudeValid && position.LongitudeValid)
{
 // Aggiorniamo l'interfaccia utente
 lblLatitude.Text = "Latitudine: " + position.Latitude;
 lblLongitude.Text =
               "Longitudine: " + position.Longitude;
 try
 {
  // Salviamo la posizione invocando il web service
  oTrackerWS.SavePosition(
      Convert.ToInt32(TrackerDevice),
      position.Latitude,
      position.Longitude);
 }
 // Gestiamo l'errore
 catch (System.Net.WebException exWeb)
 {
  string message = "";
  // Ricaviamo la risposta per estrapolare l'errore
  HttpWebResponse response =
               (HttpWebResponse)exWeb.Response;
  if (null != response)
  {
     // Estrapoliamo il messaggio di errore
     message = response.StatusDescription;
     response.Close();
  }
  lblMess.Text += "Errore nel test del WS: " +
                                       message;
 }
 catch (Exception e)
 {
   lblMess.Text = "Errore nell'invio dati!!!\nL'errore
                               è:\n" + e.Message;
 }
}
```

INSTALLAZIONE DEL SOFTWARE

Se andiamo ad effettuare la "build" della nostra applicazione (pressione dei tasti Ctrl+Shift+B) produciamo il file *TrackerMobile.exe*. Per effettuare la distribuzione dell'applicazione possiamo scegliere di creare un nuovo progetto per la creazione di un file Cabinet (CAB) in Visual Studio. Dato che, in questo caso, l'installazione dell'applicazione equivale alla copia dei file nel dispositivo (ed anche perché lo spazio a disposizione nell'articolo non è infinito), ci limitiamo a creare una cartella nel nostro dispositivo e ci copia-

mo, oltre all'eseguibile, il file di configurazione *TrackerMobile.xml* e la libreria *Microsoft.WindowsMobile.Samples.Location.dll* per l'interazione con l'apparato GPS.
Per avviare l'applicazione basta dare un colpo di pennino sull'eseguibile. Se l'apparato GPS è stato correttamente avviato e configurato, l'applicazione comincia a ricevere (ed inviare) i dati così come mostrato in **Figura 7**.

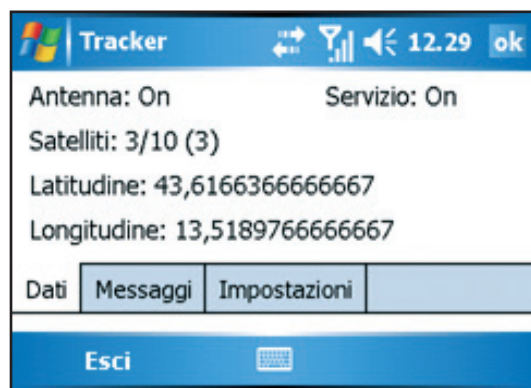



***Fig. 7:** L'applicazione TrackerMobile riceve (ed invia) i dati relativi alla posizione corrente.*

DOV'È IL MIO GPS?

Affinché funzioni tutto il meccanismo di rilevazione ed invio della posizione, dobbiamo preoccuparci di configurare correttamente l'apparato di ricezione satellitare; questo vuol dire che se utilizziamo, ad esempio, un'antenna connessa tramite Bluetooth, dobbiamo effettuare preventivamente il "pairing" dei dispositivi e definire le opportune porte COM da utilizzare per lo scambio dei dati.
La descrizione di queste operazioni esula dagli scopi dell'articolo ma è comunque importante dare alcune indicazioni sul programma di configurazione delle impostazioni GPS (Figura 8). Il programma si dovrebbe trovare nella cartella delle impostazioni di sistema del dispositivo. Il condizionale è d'obbligo dato che alcuni costruttori nascondono l'applicazione (chissà perché?) quindi potremmo anche non trovarla affatto. Per ovviare a questo inconveniente dobbiamo agire a livello del registro del dispositivo (ad esempio con il Remote Registry Editor di Visual Studio piuttosto che con uno dei tanti programmini gratuiti da installare sul dispositivo che possiamo trovare in rete).
All'indirizzo http://blogs.msdn.com/windowsmobile/archive/2006/06/07/620387.aspx possiamo trovare un dettagliato articolo che ci mostra tutte le operazioni per mostrare il programma di configurazione delle impostazioni GPS ed il suo relativo uso.

CONCLUSIONI

Lo sviluppo tecnologico di dispositivi e sistemi software ormai ci consente di realizzare applicazioni molto complesse con una relativa facilità.
Nell'articolo appena letto abbiamo avuto modo di sviluppare la prima parte di una soluzione completa per il tracciamento di dispositivi in movimento sul territorio.
Abbiamo ipotizzato una dotazione di dispositivi mobili basati sul sistema operativo Windows Mobile 5 o Windows Mobile 6, dotati di apparati per la ricezione satellitare (antenne GPS) e dotati di supporto telefonico per la comunicazione della posizione alla Centrale di Controllo.
Abbiamo quindi sviluppato sia l'applicazione mobile sia il web service a cui, la suddetta applicazione, invia i dati relativi al posizionamento. Al fine di strutturare la soluzione in più "strati logici", abbiamo altresì sviluppato una classe wrapper che incapsula la logica di persistenza delle posizioni su di una base dati. Nel caso preso in considerazione, si è scelto di usare una base dati ospitata su SQL Server 2005. Nel caso in cui scegliessimo un differente repository, potremo progettare una nuova classe wrapper senza dover toccare nessuna altra parte della soluzione.
Nella seconda parte dell'articolo ci concentreremo sull'applicazione di controllo che è basata su servizi di geolocalizzazione forniti da Google o Microsoft e sfrutta la potenzialità di AJAX per gestire il movimento delle tracce. Utilizzando quest'ultima tecnica vedremo come proprio con le tecnologie più recenti, il racciamento della posizione geografica risulterà assolutamente fluido, esattamente come lo sarebbe su un'applicazione sviluppata lato desktop e pensata per visualizzare gli spostamenti in tempo reale Questo genere di applicazione può risultare utilissima quando si tratta di tracciare il posizionamento di merci, mezzi e la dislocazione di eventuali agenti di rappresentanza

Oscar Peli

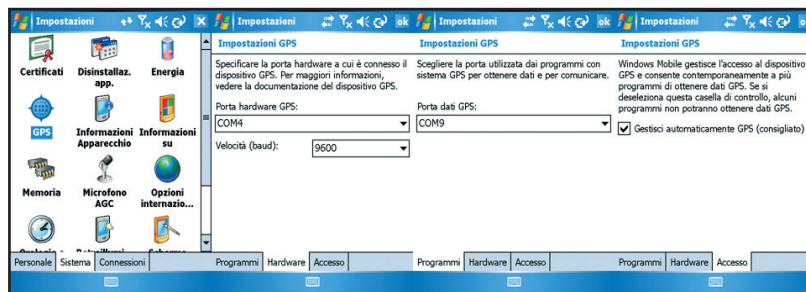



***Fig. 8:** Il programma di configurazione delle impostazioni GPS*

DOCUMENTI FUORI DI ROOT

QUANDO SI TRATTA DI CONSENTIRE IL DOWNLOAD DI FILE SOLTANTO A CHI NE POSSIEDE L'AUTORIZZAZIONE È OPPORTUNO MANTENERLI IN UNA DIRECTORY NON ACCESSIBILE DAL WEB SERVER E SPOSTARLI SOLO IN SEGUITO AD UNA RICHIESTA, VEDIAMO COME...

Conoscenze richieste

conoscenza di base del .NET Framework 2.0, ASP.NET 2.0

Software

Visual Studio 2005, .NET Framework Runtime 2.0, IIS 6 o superiore

Impegno

Tempo di realizzazione

In questo articolo affronteremo uno dei problemi a cui spesso ci si trova di fronte quando è necessario implementare un'applicazione web, fra le cui funzionalità o requisiti funzionali c'è quello di dover accedere ad un file o ad una cartella che si trova fuori, quindi al di sopra, della root del web server stesso e viceversa, quello di permettere la scrittura in una determinata cartella all'interno della root.

UN CASO REALE

Supponiamo di dover affrontare e risolvere il seguente caso di studio, per altro abbastanza reale o verosimile.

- L'azienda X ha una intranet e vuole distribuire le buste paga ai dipendenti in formato PDF all'interno della intranet stessa
- non vuole che le buste paga siano accessibili su un percorso della intranet fisico http://intranet/bustepaga/ (per evitare anche che un dipendente veda le buste paga degli altri colleghi);
- perciò il dipendente X entra in un'area riservata della intranet tramite login e password;
- richiede la busta paga del mese e gli viene inviato un codice via email;
- il dipendente immette il codice o clicca su un link contenuto nell'email ricevuta;
- la busta paga viene spostata o copiata momentaneamente da un'area esterna alla root del web ad un'area locale;
- l'utente scarica il file che a questo punto è disponibile nella intranet ad un indirizzo ben definito;
- al termine del download, oppure dopo un certo timeout, il documento viene eliminato dalla root del web per evitare che un altro dipendente possa scaricarlo.

Per realizzare un simile sistema cercheremo di utilizzare Internet Information Server ed i permessi assegnabili ad utenti per accedere ad una determinata cartella all'interno della root. L'applicazione web, scritta in ASP.NET si occuperà poi di spostare un file pdf, rappresentante la busta paga, dall'esterno all'interno della web root, e di eliminarlo dopo che l'utente lo ha scaricato o dopo la scadenza del timeout.

L'APPLICAZIONE WEB

Tralasciando gli aspetti che prescindono dallo scopo di questo articolo, e quindi per esempio l'implementazione di un sistema di autenticazione form-based, o ancora meglio l'aspetto grafico, andiamo a progettare le pagine che costituiscono la base del sistema di distribuzione delle buste.
La prima pagina necessaria è quella che permetterà ad un dipendente di richiedere la busta paga del mese inserendo i propri dati, in questo caso supponiamo che sia sufficiente il codice fiscale ed un indirizzo email sul quale ricevere un link od un codice per poter effettuare il download vero e proprio.
Il codice HTML della pagina aspx potrebbe essere il seguente:

```
<body>
  <form id="form1" runat="server">
  <div>
    Inserisci i tuoi dati per ricevere
    il codice d'accesso alla busta paga<br />
    <br />
    codice fiscale:
    <asp:TextBox ID="txtCF"
            runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator
    ID="RequiredFieldValidator1" runat="server"
                    ControlToValidate="txtCF"
ErrorMessage="RequiredFieldValidator">
*</asp:RequiredFieldValidator><br />
    email:
    <asp:TextBox ID="txtEmail" runat="server"
```

```
                  Width="334px"></asp:TextBox>
        <asp:RequiredFieldValidator
        ID="RequiredFieldValidator2" runat="server"
                    ControlToValidate="txtEmail"
ErrorMessage="RequiredFieldValidator">*
</asp:RequiredFieldValidator><br />
        <br />
        <asp:Button ID="btnRichiedi" runat="server"
       OnClick="btnRichiedi_Click" Text="Richiedi" />
                                <br /><br />
        <asp:Label ID="labError" runat="server"
              ForeColor="Red"></asp:Label></div>
    </form>
</body>
</html>
```

L'aspetto grafico della pagina sarà simile a quello mostrato nella seguente figura, niente di speciale, ma potete passarla al vostro designer grafico di fiducia per migliorarla prima di consegnarla al cliente!

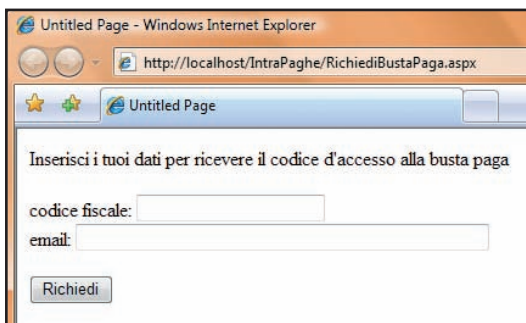



Figura 1: la pagina di richiesta delle buste

Inserendo il codice fiscale, l'email e cliccando sul pulsante Richiedi verrà eseguito il seguente codice:

```
protected void btnRichiedi_Click
                        (object sender, EventArgs e)
{
        string cf = txtCF.Text;
        string email = txtEmail.Text;
        GestoreBuste gestore =
                            new GestoreBuste();
        if (gestore.RicercaBusta(cf))
        {
            string codice=gestore.Cripta(cf);
            string link="http://"
Request.Url.Authority + Request.ApplicationPath;
            link += "/AttivaCodice.aspx?cf=" + cf +
                            "&codice=" + codice;
            new MailManager().
           SendMail("info@azienda.it",  email,
                    "Codice Busta Paga", link, "");
            labError.Text = "Il codice di accesso
                    alla busta
```

```
                    paga è stato inviato all'indirizzo
                                    email specificato";
 }
 else
 {
        labError.Text = "La busta paga richiesta non
                            è disponibile";
 }
```

La classe *GestoreBuste* in questo caso si occupa di verificare che la busta per il codice fiscale, cioè per il dipendente inserito, esista e sia disponibile.
Supponiamo infatti che le buste vengano elaborate da una società esterna, che consegna quindi i file pdf all'azienda, la quale le inserisce in una cartella al di fuori della root web. Nell'esempio esse saranno posizionate nella cartella C:\Temp\Buste.
Il metodo ricerca busta si limita a verificare l'esistenza di un file mediante il metodo File.Exists:

```
public bool RicercaBusta(string cf)
{
        try
        {
            return
                            File.Exists(Path.Combine
                    (@"C:\temp\buste",cf+".pdf"));
        }
        catch (Exception ex)
        {
            throw ex;
        }
}
```

Se il test è positivo, il codice fiscale viene criptato, per poter salvare il file pdf con un nome differente dal codice fiscale stesso, evitando che un qualunque dipendente, conoscendo il codice fiscale di un collega, tenti di scaricarne la busta paga, una volta che questa si trovi nella web root.
Il metodo Cripta usato nel codice in allegato non fa nient'altro che spostare le lettere avanti di una:

```
public string Cripta(string cf)
{
        StringBuilder sb = new StringBuilder();
        sb.Append(cf.ToLower());
        for (int i = 0; i < cf.Length; i++)
        {
            if (sb[i] == 'z')
                sb[i] = 'a';
            else sb[i] = (char)(sb[i] + 1);
        }
```

NOTA

INVIO EMAIL

Per permettere l'invio dell'email contenente il codice di attivazione è necessario modificare il file di configurazione web.config inserendo i dati del vostro server SMTP e magari i dati di accesso di un account, esattamente all'interno della sezione appSettings, impostando i value delle chiavi SmtpServer, SmtpPort e csì via.

```
        return sb.ToString();
}
```

Per utilizzi reali sarebbe meglio utilizzare funzioni di crittografia reali e sicure. Subito dopo, con il codice criptato, viene creato un link alla pagina per attivare il codice di download stesso, ed esso viene inviato per email al richiedente (anche questo: per un utilizzo reale non si permetterebbe l'inserimento dell'indirizzo, ma verrebbe utilizzato quello del dipendente corrispondente al codice fiscale inserito).

ATTIVARE IL CODICE

Il link costruito nella precedente sezione permette di attivare il codice e richiedere dunque lo spostamento del file pdf (rappresentante la busta paga) da un'area esterna alla web root, all'interno di una cartella raggiungibile via intranet o via web.

Naturalmente è necessario configurare i permessi di tale cartella in maniera precisa, come mostrato nel prossimo paragrafo, che costituisce il nucleo dell'articolo.

```
protected void Page_Load
                    (object sender, EventArgs e)
{
        if (!IsPostBack &&
                Request.QueryString["cf"] !=
    null && Request.QueryString["codice"] != null)
        {
            string cf = Request.QueryString["cf"];
            string codiceCryptato =
                    Request.QueryString["codice"];
            string codice = new
            GestoreBuste().Decripta(codiceCryptato);
            if (cf == codice)
            {
                labMessage.Text="Clicca sul link in
                    basso per scaricare la busta paga";
                new GestoreBuste().
                    CopiaBustaPaga(cf, codiceCryptato);
                lnkBtDownload.CommandArgument =
                            codiceCryptato; ;
                lnkBtDownload.Visible = true;
                timer = new Timer(3 * 60 * 1000);
               //busta  paga disponibile per 3 minuti;
                timer.Elapsed += new
                 ElapsedEventHandler(timer_Elapsed);
                timer.Enabled = true;
            }
            else
            {
                labMessage.Text =
                        "Errore, codice non valido";
                lnkBtDownload.Visible = false;
            }
        }
}
```

APPLICATION POOLING

La versione 5.0 di Internet Information Server utilizza ASP.NET process model eseguendo (Aspnet_wp.exe). Mediante tale modello, ogni versione unica di un'applicazione esegue automaticamente in un processo separato a sé dedicato. Tutte le applicazione che utilizzano la stessa versione del runtime di .NET condividono lo stesso processo (oppure un gruppo di processi nella modalità chiamata Web garden). Dalla versione 6.0 è stato introdotto un nuovo modello di esecuzione (w3wp.exe) ed una nuova funzionalità di isolamento detta application pooling. Quest'ultima consente a diverse applicazioni di eseguire insieme in uno o più processi condivisi. Al contrario, le applicazioni assegnate ad application pool differenti non verranno mai eseguite all'interno di uno stesso processo.

Nel *Page_Load* della pagina viene verificata la corrispondenza del codice fiscale e del codice criptato, e solo in caso affermativo, la busta paga corrispondente viene copiata dall'area esterna alla root ad una cartella interna, mediante il metodo *CopiaBustaPaga* della classe *GestoreBuste*:

```
public void CopiaBustaPaga(string cf,string codice)
{
        try
        {
            string destPath =
HttpContext.Current.Server.MapPath(@"~\buste");
            FileInfo fi = new
              FileInfo(@"C:\temp\buste\"+cf+".pdf");
            fi.CopyTo(Path.Combine(destPath,
                            codice+".pdf"),true);
        }
        catch (Exception ex)
        {
            throw ex;
        }
}
```

La busta viene copiata all'interno della directory buste della web root. Subito dopo il LinkButton viene configurato impostando la proprietà CommandArgument con il codice criptato. Dato che non si vuol tenere la busta disponibile nella web root per un tempo indefinito, ed inoltre non è determinabile quando e se il dipendente ha scaricato correttamente la propria busta, viene utilizzato un timer per rendere la busta disponibile per un tempo fissato a 3 minuti, trascorsi i quali il file viene eliminato ed il

link er il download disattivato:

```
void timer_Elapsed(
          object sender, ElapsedEventArgs e)
{
        new
```

Untitled Page - Windows Internet Explorer
lhost/IntraPaghe/AttivaCodice.aspx?cf=pllntn75d
Untitled Page

Area riservata

Clicca sul link in basso per scaricare la busta paga

Scarica la tua busta

Figura 2: Pagina per il download della busta paga

```
        GestoreBuste().EliminaBustaPaga(ViewState
                              ["BustaPaga"] as string);
        lnkBtDownload.CommandArgument = null;
        timer.Enabled = false;
        timer.Elapsed -= timer_Elapsed;
}
```

La pagina di attivazione e download è mostrata nella figura 2.

SCARICARE LA BUSTA PAGA

Il download della busta paga, viene effetuato in maniera piuttosto semplice e classica, modificando le intesazioni della risposta http, il content type, e scrivendo dunque il contenuto del pdf stesso:

```
    protected void lnkBtDownload_Click
                  (object sender, EventArgs e)
    {
        string codice =
              lnkBtDownload.CommandArgument;
        string file = Server.MapPath
                  ("~/buste/" + codice + ".pdf")
        if (file == null || !File.Exists(file) )
        {
            labMessage.Text =
            "La busta paga non è più disponibile";
        }
        else
        {
            Response.Clear();
            FileInfo fi = new FileInfo(file);
            Response.AddHeader
                  ("Content-Disposition",
             "attachment; filename=bustapaga.pdf");
            Response.AddHeader("Content-Length",
                              fi.Length.ToString());
            Response.ContentType =
                  "application/octet-stream";
            Response.WriteFile(fi.FullName);
            Response.End();
        }
    }
```

Nel caso in cui fosse scaduto il timeout impostato nel precedente *Page_Load*, la proprietà *CommandArgument* del *LinkButton* di download sarà null, e naturalmente non esisterà più neanche il file pdf all'interno dela web root. Quindi verrebbe visualizzato un messaggio di errore, ad indicare che la busta paga richiesta non è più disponibile.

CONFIGURAZIONE PERMESSI

Le pagine ASP.NET sono eseguite all'interno di un preciso processo di sistema, cioè di un programma Windows. Come saprete, ogni programma di Windows viene eseguito sotto una specifica identità. Per default, in particolare, il processo ASP.NET esegue sotto una identity predefinita, ma può anche essere utilizzata la tecnica di impersonation per far eseguire il processo sotto una differente identità.

Per incrementare la sicurezza dell'applicazio-

DIRECTORY TRAVERSAL

La cosiddetta vulnerabilità di directory traversal può verificarsi in all'interno di un qualunque web server o a causa di un'applicazione web non adeguatamente protetti o configurati. Nel caso di applicazioni web dinamiche, è possibile passare dei parametri ad un pagina utilizzando per esempio l'indirizzo digitato nel browser, cioè, in parole più tecniche, una QueryString:

http://www.miosito.com/readnews.aspx?file=news1.html

Nell'esempio precedente la pagina richiesta dal browser è chiamata readnews.aspx ed inoltre viene spedit al web server un parametro file con valore pari al nome di un file da visualizzare news1.html.

La pagina, una volta eseguita, tenta di leggere il file indicato e quindi di visualizzarlo nel browser.

Un possibile utente malintenzionato potrebbe immediatamente avere l'idea di utilizzare una possibile vulnerabilità, modificando manualmente il nome del file da leggere, crando ad hoc un nuovo indirizzo, come il seguente:

http://www.miosito.com/readnews.aspx?file=../../../../WINNT/win.ini

In questa maniera tenta di risalire lungo l'albero delle cartelle per andare a trovare il file win.ini, ma naturalmente (si spera!) un web server configurato correttamente non permetterà di risalire l'albero uscendo fuori dalla web root.

ne ASP.NET è quindi necessario assicurarsi che il processo ASP.NET esegua sotto una identità che abbia dei privilegi minimi di accesso al file system, o comunque ben definiti e solo quelli necessari per eseguire l'applicazione stessa.
Nel nostro caso è necessario fare in modo che il processo abbia privilegi di accesso e scrittura ad una determinata directory, quella che conterrà le buste paga, e naturalmente evitare che un

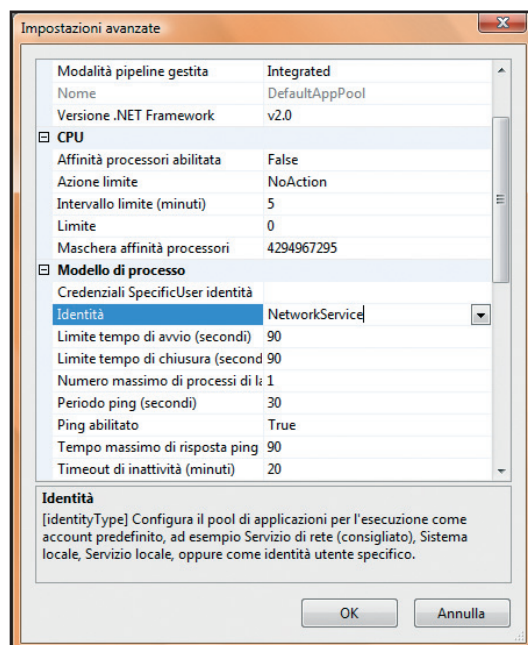



Figura 3: Identità dell'application pool

utente qualunque possa accedervi, anche in sola lettura, semplicemente digitando un indirizzo nella barra del browser.
Tutto ciò serve a ridurre anche la possibile vulnerabilità dell'intero sistema.
Su un server web dotato di Internet Information Services (IIS) 6.0, il processo ASP.NET esegue all'interno dell'application pool per l'applicazione web. Tale application pool assegna al processo ASP.NET una precisa identità, che per default è NETWORK SERVICE o SERVIZIO DI RETE nella versione in italiano.
Nella versione 5.0 di IIS invece, fornita con i sistemi operativi Windows 2000 e XP Professional, ASP.NET viene eseguito da un cosiddetto worker process, è l'utente sotto cui esegue è proprio l'utente del processo aspnet_wp.exe (che per default è l'account ASPNET).
Bisogna dare permessi di scrittura all'utente NETWORK SERVICE, per far ciò basta fare clic con il tasto destro sulla cartella che conterrà le buste baga dei dipendenti, all'interno della root del web server IIS, per esempio C:\inetpub\wwwroot\IntraPaghe\buste, e scegliere la voce Proprietà dal menu contestuale.

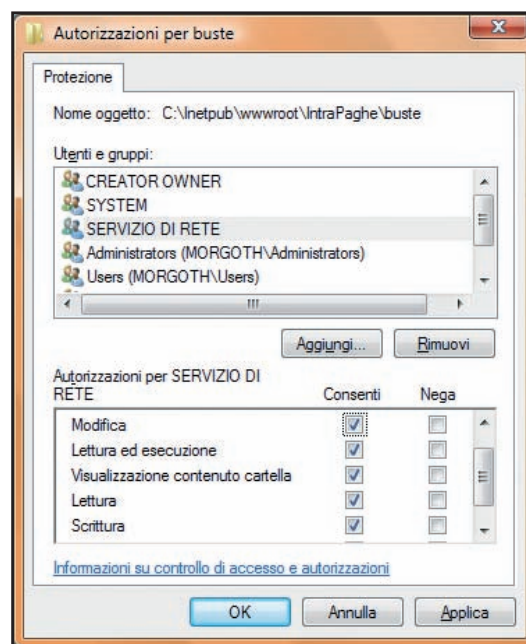



Figura 4: Modifica dei permessi per l'utente SERVIZIO DI RETE

Dalla finestra Proprietà, bisogna ora selezionare l'etichetta Protezione e quindi fare lic su Modifica per modificare i permessi della cartella.
Dalla finestra autorizzazioni, è possibile aggiungere un utente e determinarne i permessi.
In questo caso basta fare clic su Aggiungi, inserire il nome utente NETWORKSERVICE o SERVIZIO DI RETE e quindi dare all'utente stesso permessi di Modifica, e quindi Lettura e Scrittura, come mostrato nella figura seguente.

CONCLUSIONI

In questo articolo abbiamo mostrato come implementare un servizio web per distribuire le buste paga ai dipendenti di una azienda, nella intranet, evitando di conservare i file all'interno della web root, ed anzi copiandole al suo interno su richiesta del dipendente, e solo in seguito alla corretta verifica di un codice di richiesta. Inoltre si è mostrato come configurare i permessi in maniera adeguata, esponendo anche i relativi concetti di security. In casi di implementazione reale sarà opportuno prima di tutto creare un meccanismo di autenticazione degli utenti sufficientemente sicuro, inoltre sarà necessario proteggere anche le directory che risultano al di fuori del web server in modo opportuno onde evitare che qualche utente possa, inviando query string ad hoc, risalire l'albero delle directory fisiche del server

Antonio Pelleriti

L'AUTORE

Antonio Pelleriti, ingegnere informatico, sviluppa software da più di dieci anni e si occupa di .NET sin dalla prima versione Beta. È cofondatore e chief software architect di DynamiCode©® (www.dynamicode.it), software factory che utilizza principalmente .NET per la progettazione e lo sviluppo software, può essere contattato per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@dotnetarchitects.it

AUTENTICAZIONE E RUOLI CON PHP

NELLA PROGETTAZIONE DI UN'AREA RISERVATA PER IL WEB, LA PRIMA COSA A CUI SI DEVE PENSARE È: "CHI FA COSA?". VEDIAMO COME SI POSSONO GESTIRE "ACCESSI E RUOLI" UTILIZZANDO UNA LIBRERIA GRATUITA DELLE CLASSI PEAR

Nel costruire un sito o un'applicazione internet, le politiche di sicurezza necessitano delle la dovuta attenzione. La loro gestione implica decisioni sul controllo degli accessi, individuando una classificazione in tipologie degli utenti per cui si identificano dei ruoli ma anche le operazioni consentite e quelle bloccate, a cui non è possibile accedere se l'utente manca di autorizzazione. Ad esempio posto che l'utente X abbia l'autorizzazione ad accedere al sito, bisognerà decidere se questo utente ha i permessi di amministratore oppure no. È una situazione classica, visibile nei blog ad esempio, ma in realtà in qualunque sito che necessiti un'autenticazione.
Il seguente articolo mostrerà come implementare la sicurezza per sistemi sviluppati in *PHP*, usando il package *LiveUser*, contenuto nella collezioni di classi *Pear*.

PHP E CLASSI PEAR COSA SONO ?

Con un obiettivo molto simile a quello perseguito da *CPAN* per gli sviluppatori Perl, Pear sta per **PHP Extension and Application Repository**. L'idea alla base è quella di fornire ai PHP developers pacchetti (circa 440) di classi open-source pronte da usare, garantendo un sistema di distribuzione del codice e definendo e promuovendo nel contempo uno standard di programmazione del linguaggio. Ogni package di codice PEAR realizza un progetto indipendente sotto l'egida del grande progetto iniziale, con un suo team di sviluppo, uno per il controllo delle versioni e uno per la documentazione. Ogni package è costituito da codice sorgente o da codice binario, o, in alcuni casi. *La Library of PHP Code*, che è parte più corposa del progetto, è di tipo **Structured** in quanto, trattandosi di una libreria di classi, sfrutta abbondantemente le possibilità di riutilizzo del codice concesse dall'approccio **Object Oriented** attraverso l'ereditarietà o l'associazione. Tutte le classi derivano direttamente o indirettamente dalla **classe base PEAR**, contenuta nel file PEAR.php. Nell'eventualità tali dipendenze ci fossero, come nel caso, per esempio di *PEAR MDB*, che necessita di *XML_PARSER* e della classe base Pear in versione lmeno 1.0.b1, queste vengono indicate in un apposito box: *Dependencies*.

COME INIZIARE ?

La prima cosa da fare è installare LiveUser, che risiede all'interno di Pear. Al fine di eseguire correttamente l'operazione e non avere intoppi, per prima cosa bisogna far partire Pear, dopodiché lanciamo il seguente comando:
pear install LiveUser
Nel caso sia stata scaricata una nuova release del pacchetto da link http://pear.php.net/, esso può essere installato localmente. Per fare questo è necessario scrivere il seguente comando sulla shell:
$ pear install LiveUser.tgz
Questo comando installerà automaticamente il package e non richiede la connessione ad internet.

LIVEUSER: LA CLASSE CHE FA PER NOI

Live User è la collezione di classi per la gestione dell'autenticazione degli utenti e degli accessi. La sua struttura è basata su dei contenitori: un concetto molto simile a quello del driver. Per utilizzare un qualunque device abbiamo bisogno di un driver specifico alla marca e al modello del device stesso, così come per utilizzare LiveUser per determinate operazioni, abbiamo bisogno del giusto contenitore per quella particolare operazione. Il design del package si basa su due contenitori: uno per le autorizzazioni, uno per i permessi. Poi c'è la classe LiveUser che gestisce i *check* e attiva il funzionamento di uno o l'altro contenitore. Vediamo come: al momento dell'inserimento dei dati da parte di un utente, la classe LiveUser va a ricercare, all'interno di ciascun contenitore i dati memorizzati fin

quando non trova l'utente: questo significa che i contenitori hanno al loro interno tutti i dati degli utenti ed è compito della classe definire il contenitore in cui il dato utente debba essere ritrovato.

DUE CONTENITORI PERCHÈ?

Un esempio servirà a chiarirci le idee. Supponiamo di avere un'applicazione internet che può essere usata sia come strumento interno all'azienda, accessibile dai dipendenti, sia da utenti esterni all'azienda che vi si connettono attraverso Internet.
Ovviamente, le operazioni consentite a ciascun gruppo saranno differenti. Per i primi, non c'è bisogno di crearsi un account, in quanto sarà stato precedentemente creato dall'amministratore del sistema. I secondi, invece, se vogliono accedere a determinate informazioni, devono registrarsi prima come utenti, creando un nuovo account che sarà memorizzato nel database locale della macchina che fa da web server. Il contenitore relativo alle autorizzazioni conterrà i dati degli account di entrambi gli utenti.

Definendo il contenitore dei permessi, è possibile gestire i diritti di ciascun utente, differenziandoli comunque in base alla tipologia. Basta, per centrare l'obiettivo, definire uno schema di permessi per l'applicazione. Lo split in differenti contenitori permette di integrare facilmente nuove applicazioni con vecchie, ognuno delle quali potrà essere resa accessibile o meno a seconda della categoria di accesso che gli viene attribuita. I permessi altro non sono che un ID numerico e una stringa identificativa. Quando è necessario capire a quale gruppo appartiene l'utente X che ha fatto il login, viene letto nel contenitore dei permessi l'ID che gli è stato associato e che viene propagato per tutta la sessione. L'associazione dell'ID numerico alla stringa che ne identifica un *nome proprio* evita la memorizzazione della corresponsione del numero con il permesso concettualmente associato, oltre che ad agevolare l'eventuale trasferimento da un server all'altro dell'applicazione che usa LiveUser, prevedendo che qualche altra applicazione possa utilizzare lo stesso numero di permesso che, una volta identificato da una costante, non può generare confusione.

Classe	Descrizione
Class: LiveUser	Classe principale del package per gestire un sistema di login utente. Essa crea un oggetto LiveUser, si occupa del login e memorizza l'oggetto LiveUser all'interno della sessione.
Class: LiveUser_Auth_Common	Questa classe fornisce un insieme di funzioni per l'implementazione di un sistema di autorizzazioni utente per website in hosting. Tutte le autorizzazioni di tipo backends/containers devono essere ereditate da questa classe base.
Class: LiveUser_Auth_DB	PEAR::DB backend driver per la classe LiveUser. Un oggetto di tipo PEAR::DB può essere passato al costruttore per riutilizzare una connessione esistente. Altrimenti, può essere passato un DSN per aprirne una nuova.
Class: LiveUser_Auth_MDB2	PEAR::MDB2 backend driver per la classe LiveUser.
Class: LiveUser_Auth_PDO	PEAR::MDB2 backend driver per la classe LiveUser.
Class: LiveUser_Auth_PEARAuth	PEAR::Auth backend driver per la classe LiveUser. Le opzioni di setup della classe possono essere passate al costruttore. Scegliere il giusto contenitore di auth e le opzioni permette di settare 'container' e 'options' rispettivamente nell'array di memorizzazione.
Class: LiveUser_Auth_Session	Contenitore di sessione per l'autenticazione
Class: LiveUser_Auth_XML	Driver XML per l'autenticazione
Class: LiveUser_Misc_Schema_Install	Classe per l'istallazione dello schema di database
Class: LiveUser_Perm_Complex	Container complesso per la gestione dei permessi
Class: LiveUser_Perm_Medium	Container medio per la gestione dei permessi
Class: LiveUser_Perm_Simple	Classe base per la gestione dei permessi. Fornisce il più semplice set di permessi, dove ogni permesso viene consentito a ogni utente.
Class: LiveUser_Perm_Storage	Classe di astrazione per tutti i containers di memorizzazione
Class: LiveUser_Perm_Storage_DB	DB container per la gestione dei permessi
Class: LiveUser_Perm_Storage_MDB	MDB container per la gestione dei permessi
Class: LiveUser_Perm_Storage_MDB2	MDB2 container per la gestione dei permessi
Class: LiveUser_Perm_Storage_PDO	PDO container per la gestione dei permessi.
Class: LiveUser_Perm_Storage_SQL	SQL container per la gestione dei permessi.
Class: LiveUser_Perm_Storage_XML	XML container per la gestione dei permessi.

Tabella 1: Un elenco delle classi di liveuser con relativa descrizione

LE CLASSI DEFINITE IN LIVEUSER

Una breve descrizione delle classi LU ci permetterà di scoprire le funzionalità che ci vengono proposte dal package. Sono presenti non solo la classe principale LiveUser, ma anche le classi per la gestione delle autorizzazioni sia in locale sia in *backend*. Abbiamo già detto che lavora con i contenitori: ve ne sono per l'autenticazione e per i permessi, per i quali esistono appositi costruttori. La gestione dei permessi con LiveUser è demandata al package *LiveUser_Admin*, che deve essere installato anch'esso per poterne fare uso. Esso è composto di tutte le classi necessarie per amministrare i permessi e permette di aggiungere, modificare, cancellare e visualizzare permessi, implicazioni di permessi, utenti, gruppi, aree, applicazioni e supporta i contenitori *PEAR::DB, PEAR::MDB, PEAR::MDB2, PECL::PDO*. Una descrizione più o meno completa delle classi che compongono il package è contenuta nella tabella che proponiamo nella pagina seguente.

AUTENTICAZIONE E PERMESSI

quando si utilizzano dei database, è necessaria una preventiva autenticazione al fine di controllare che gli accessi ad esso siano consentiti. L'autenticazione consiste nel riconoscere l'utente per essere sicuri che esso sia esattamente chi dice di essere. Il modo più comune di implementarla consiste nel meccanismo di login\password: si chiede all'utente di specificare il proprio username e dopo di verificarlo mediante una parola chiave che si presuppone sia l'unico a conoscere. Nelle applicazioni web, in cui c'è una continua interazione tra client e server, è necessario che ogni volta che viene costruita una pagina secondo le specifiche dell'utente, si tenga traccia del fatto che esso sia stato preventivamente riconosciuto e dunque che possa fruire delle informazioni che si stanno per visualizzare. Di solito questo si fa impostando una variabile di sessione che viene testata in ogni pagina da visualizzare per la verifica dell'utente e che si perde quando viene chiuso il browser, dopo un certo periodo di inutilizzo della sessione stessa, oppure mediante il logout. Alcune policies preservano l'intero processo: la lunghezza minima e i caratteri che compongono la password che, per motivi di sicurezza, è opportuno sempre che sia costituita da un insieme combinato e vario di lettere, numeri e caratteri di interpunzione o simboli; un meccanismo di blocco nel caso di reiterati tentativi di login fallimentari, un limite di durata degli account non utilizzati; l'opportunità di crittografare le password all'interno del database per evitare furti ed attacchi. I permessi, invece, sono un modo per garantire un accesso al database che non solo sia controllato, ma anche ristretto a determinati usi: essi garantiscono che le operazioni che può fare l'utente siano esclusivamente quelli che ad esso sono concessi. Ovviamente, questo non è il caso in cui un singolo utente utilizza il desktop: in questo caso basta proteggere il computer con username e password per l'accesso. Di solito, per la gestione dei permessi, è necessario predisporre nel database una tabella che prevede i seguenti elementi: la lista delle funzioni del sistema, una lista di tutte le persone a cui è consentito l'accesso al sistema, le quali si autenticano mediante username e password, nei modi specificati precedentemente; una lista di tipologie di accessi che identifichino quali funzioni sono accessibili a ciascun utente, sia esso singolo che inteso come gruppo e le operazioni che sono ad esso consentiti: solo visualizzazione, modifica, cancellazione, inserimento.

UN ESEMPIO: IL GIORNALE ON-LINE

Implementiamo una semplice applicazione per capire come funziona il package.
Supponiamo che essa sia creata per permettere di pubblicare news su una web page. Ci sono tre parti: Cronaca, Finanza e Gossip: la lettura delle news non è riservata e dunque accessibile a chiunque. Le aree di inserimento e modifica ovviamente sono ristrette e c'è bisogno di un login per poter operare.
Per completezza, abbiamo inserito anche la descrizione di *index.php* che si occupa di gestire il database. Il seguente codice è inserito nel file *admin.php*.
Dopo aver richiamato le classi che serviranno alla creazione dei contenitori per autorizzazioni e permessi, utilizzando *require_once* che include e valuta i file specificati durante l'esecuzione dello script permettendo di distinguere la situazione in cui il file è stato già incluso al fine di evitare una nuova, successiva inclusione, stabiliamo una connessione al dsn, che conterrà tutte le informazioni per l'autenticazione. Di seguito, creiamo un nuovo oggetto- contenitore di tipo *LiveUser_Admin_Auth_Container_DB* con dati di autorizzazione del nuovo utente: questa classe, come la successiva, necessaria per i permessi, si trova nel package *LiveUser_Admin*, di cui abbiamo già detto prima che si occupa della di creazione dei permessi.

```
<?php
require_once 'config.inc.php';
require_once
    'LiveUser/Admin/Perm/Container/DB_Medium.php';
require_once
            'LiveUser/Admin/Auth/Container/DB.php';

$lu_dsn = array('dsn' => $dsn);
$objRightsAdminAuth = new
    LiveUser_Admin_Auth_Container_DB(
        $lu_dsn, $conf['authContainers'][0]
    );
$objRightsAdminPerm = new
  LiveUser_Admin_Perm_Container_DB_Medium($lu_d
                                    sn, $conf);
```

Controlliamo se il modulo in backend è inizializzato correttamente; se questo è verificato, aggiungiamo il nuovo utente, iniziando col richiamare il metodo che si occupa di questo per le autorizzazioni, *addUser* , i cui parametri sono username e password e il terzo valore impostato a *TRUE* per indicare che all'utente bisogna attribuire dei permessi. Testiamo non ci siano errori in relazione alla variabile *$user_auth_id* e proseguiamo aggiungendo l'utente anche al conteni-

tore dei permessi.

```
if (!$objRightsAdminPerm->init_ok) {
    die('Impossibile inizializzare' .
                $objRightsAdminPerm->getMessage());
}

$objRightsAdminPerm->setCurrentLanguage('EN');

$user_auth_id = $objRightsAdminAuth-
            >addUser('Patrizia', 'IoProgrammo', true);

if (DB::isError($user_auth_id)) {
    $user_auth_id->getMessage();
}

$objRightsAdminPerm->addUser($user_auth_id);

echo '$user_id created ' . $user_auth_id . "n";
```

Le righe di codice che seguono si occupano di suddividere in aree la nostra applicazione: innanzitutto inseriamo tutta la nostra applicazione nella gestione dei permessi, in quanto l'amministratore, ovviamente, può agire in tutte le aree e poi specifichiamo l'area "*Area*", aggiungendola mediante *addArea*, il cui primo attributo è *$app_id*, che identifica l'applicazione in oggetto.

```
$app_id = $objRightsAdminPerm-
            >addApplication('LIVEUSER', 'website');
$area_id = $objRightsAdminPerm-
    >addArea($app_id, 'AREA', 'la nostra unica area');
```

Prima abbiamo creato i contenitori delle autorizzazioni e permessi, poi l'area che sarà gestita dai permessi: occupiamoci ora della creazione dei permessi stessi. Ve ne saranno due: uno per la modifica delle news, l'altro per scrivere. Per fare ciò utilizziamo *addright*, specificando, come primo parametro, l'area di riferimento che abbiamo creato precedentemente.

```
$right_1 = $objRightsAdminPerm-
        >addright($area_id, 'MODIFICA NEWS',   'Leggi
                                    qualcosa');
$right_2 = $objRightsAdminPerm-
            >addright($area_id, 'SCRIVI NEWS',  'Scrivi
                                    qualcosa');
echo 'Created two rights with id ' . $right_1 . ' and '
                                . $right_2 . "n";
```

I diritti creati sono due: le seguenti linee di codice si preoccupano dell'attribuzione.

```
$objRightsAdminPerm-
            >grantUserRight($user_auth_id, $right_1);
$objRightsAdminPerm-
            >grantUserRight($user_auth_id, $right_2);
                                            ?>
```

Dopo aver richiamato le classi che ci serviranno, creiamo una connessione con il database e costruiamo un nuovo template per la visualizzazione dei dati, richiamando index.tpl. Per la compilazione del template abbiamo usato *HTML_Template.it*, soluzione PEAR appositamente creata: il motore di template si preoccupa di leggerli, compilarli e crearne un script php. Esso fornisce un modo semplice di separare la logica e il contenuto dell'applicazione dalla sua presentazione, secondo la regola che vuole il programmatore essere figura diversa dal proget-

IL FILE DI TEMPLATE

Per completezza riportiamo index.tpl, che si preoccupa di costruire il form di inserimento dati e di richiamare index.php. Il template si preoccupa di visualizzare le notizie contenute nel database e di fare il login per la modifica e l'inserimento di nuovi articoli.

```
<html>
<head>
    <title> LiveUser</title>
     <style type="text/css"
            media="screen">@import
        "layout_frontend.css";</style>
</head>

<body>
    <h1>LiveUser</h1>
    <p>Un esempio di utilizzo delle
        classi Pear per il controllo degli
      accessi utente usando PHP</br>
    <p>Di Caterina Patrizia
                        Morano</p>
</div>
<div class="content">
<h2>Cronaca:</h2>
{CRONACA }
</div>
<div class="content">
<h2>Finanza:</h2>
{FINANZA}
</div>
<div id="navAlpha">
<h2>Gossip :</h2>
{GOSSIP}
</div>
<div id="navBeta">
<h2>Login Amministratore</h2>
<!-- login form to the admin part -->
<form method="post"
            action="admin.php">
<table style="border:1px dashed
                    black;">
    <tr>
        <td>login:</td>
        <td><input name="handle"
                type="text" size="5"
            manlength="15" /></td>
    </tr>
    <tr>
        <td>password:</td>
        <td><input name="passwd"
            type="password" size="5"
            manlength="10" /></td>
    </tr>
    <tr>
        <td>Remember me <input
                type="checkbox"
        name="rememberMe" /></td>
    </tr>
    <tr>
        <td colspan="2"><input
type="submit" value="Log-in"></td>
    </tr>
</table>
</form>
<p>
<h2>Flash:</h2>
{FLASH}
</p>
</div>
</body>
</html>
```

tista di applicazioni. Osserviamo, dunque, che le inclusioni ne prevedono il richiamo e le linee successive l'inizializzazione dopo, ovviamente, aver effettuato la connessione al db.

```
<?php
require_once 'DB.php';
require_once 'config.inc.php';
require_once 'HTML/Template/IT.php';

$db  = DB::connect($dsn);
$tpl =& new HTML_Template_IT('./');

$tpl =& new HTML_Template_IT('./');
$tpl->loadTemplatefile('index.tpl', true, true);
```

Ovviamente è necessario associare i contenuti a ciascuna variabile: lo facciamo utilizzando *setVariable*, che a database aperto, va a ricercare le notizie e le recupera per la visualizzazione.

```
$tpl->setVariable('CRONACA', getNews($db,
                                    'cronaca'));
$tpl->setVariable('FINANZA',  getNews($db,
                                    'finanza'));
$tpl->setVariable('GOSSIP',   getNews($db,
                                    'gossip'));
$tpl->setVariable('FLASH',      getNews($db,
                                    'flash'));
$tpl->show();
```

Vediamo ora come tirar fuori dal database le notizie: ciascuna di esse è caratterizzata dall'avere un *id*, una data di inserimento, un titolo e il contenuto, la *query* utilizzata in *getAssoc* richiede tutti questi elementi.
Ricordiamoci la gestione degli errori..

```
function getNews(&$db, $newsCategory)
{
    $query = "
        SELECT
            news_id      AS id,
            news_date    AS date,
            news_title   AS title,
            news_content AS content
        FROM
            news
        WHERE
            news_category = '$newsCategory'";
    $news = $db->getAssoc($query,
                              DB_FETCHMODE_ASSOC);
    if (DB::isError($news)) {
        die($news->getMessage() . ' ' . $news-
                                   >getUserinfo());
    } else {
        $tpl =& new HTML_Template_IT('./');
```

L'ultima riga si è preoccupata di creare un nuovo oggetto di tipo *HTML_Template_IT()*, che è poi memorizzato in una directory diversa per gli script che lo usano. Le righe che seguono invece, richiamo il metodo *loadTemplateFile()* usato per caricare il file del template nel suo motore. Il file del template contiene i blocchi e le variabili che il motore rimpiazza con i dati attuali: esso li cerca nel file nella directory specificata dal costruttore. Una volta che il template è caricato, è necessario assegnare i valori alle variabili: di questo si occupa il metodo *setVariable()*, mentre *setCurrentBlock()* ricorda al motore i blocchi all'interno dei quali cercare le variabili; *parseCurrentBlock()* rimpiazza le variabili con il loro valore.

```
        $tpl->loadTemplatefile('news.tpl', true, true);
        foreach($news as $name) {
            foreach($name as $cell) {
                $tpl->setCurrentBlock('cell');
                $tpl->setVariable("DATA", nl2br($cell));
                $tpl->parseCurrentBlock('cell');
            }
            $tpl->setCurrentBlock('row');
            $tpl->parseCurrentBlock('row');
        }
        return $tpl->get();
    }
}?>
```

SE NON ABBIAMO BISOGNO DI PERMESSI

Abbiamo detto che la classe LiveUser è quella che lo sviluppatore usa per testare l'associazione autenticazione\permesso. Essa si interfaccia con admistration classes, chi scrive il codice interagisce solo con LiveUser: ovviamente, nel caso in cui dobbiamo fare una semplice autenticazione non c'è bisogno di LiveUser ma basta Pear::Aut che svolge eccellentemente il suo compito senza complicazioni che in questo caso non avrebbero nessuna utilità pratica. è inoltre disponibile un altro pacchetto, da utilizzare insieme: LiveUser_Admin. Esso permette di amministrare i dati usati da LiveUser e in particolare, permette di fare operazioni di inserimento, modifica, cancellazione e visualizzazione su diritti, utenti, gruppi di utenti, sottogruppi, suddivisione in aree dell'applicazione, le applicazioni e diritti a cui si accede in conseguenza dell'accesso di altri diritti consentiti.

CONCLUSIONI

utilizzando LiveUser con qualche semplice accorgimento è possibile facilmente implementare una gestione completa di autorizzazioni e ruoli senza per questo dover scrivere da zero il proprio codice

Caterina Patrizia Morano

CREARE REPORT EXCEL DAL WEB

ESPORTARE I DATI DI OUTPUT DALLE NOSTRE APPLICAZIONI IN COMODI FILE EXCEL È SEMPRE STATO MOLTO UTILE SE NON ESSENZIALE. CON LE GIUSTE LIBRERIE ORA È ANCORA PIÙ FACILE. UTILIZZIAMONE UNA SEMPLICE E GRATUITA

È capitato a tutti, prima o poi, nella propria carriera di sviluppatori di sentirsi chiedere dal cliente se tutte quelle bellissime griglie dinamiche e colorate potessero essere in qualche modo esportate in un formato più “pratico” per permettere di elaborare i dati e generare diversi tipi di statistiche o grafici. Come non pensare quindi immediatamente al tanto diffuso ed apprezzato *Microsoft Excel*. Ecco quindi che subito ci si ritrova sul web alla ricerca di qualcosa che ci possa permettere di esportare in formato Excel. I più temerari, sapendo che Excel gestisce un formato chiamato *XML Spreadsheet*, si potranno cimentare nella generazione manuale di questo tipo di file sulla base dei dati forniti dalla loro applicazione, mentre i più andranno alla ricerca di una libreria o componente che gli permetta di creare documenti Excel più facilmente senza doversi necessariamente studiare il formato *XML Spreadsheet*. Per questi ultimi sono disponibili diverse librerie atte a questo scopo e sebbene la maggior parte di queste siano prodotti commerciali, anche in questo caso non mancano soluzioni gratuite di livello più che buono. È il caso della libreria *CarlosAg Excel Xml Writer Library*, sviluppata completamente in C# e totalmente gratuita. In questo articolo impareremo ad utilizzare questa libreria per produrre un foglio di lavoro Excel di media complessità generandolo dall'interno di una pagina Web anche se la libreria può essere tranquillamente utilizzata da un'applicazione Windows Forms o Console. L'unica limitazione della libreria che utilizzeremo riguarda il fatto di non poter creare dei grafici all'interno del foglio di lavoro Excel ma considerato che tutto il resto è possibile, questa è una limitazione trascurabile.

INSTALLAZIONE

Partiamo allora con il download della libreria dal sito del suo autore e ricordiamoci di effettuare il download insieme ad essa, anche dell'help in formato CHM (formato Help di Microsoft) e dell'ottimo *Code Generator* che ci permetterà di generare più facilmente il nostro codice e di cui parleremo più avanti. La libreria in realtà non necessita di una vera e propria installazione in quanto è sufficiente referenziare nel nostro progetto (Web, Console o WinForms) la relativa DLL per utilizzarla immediatamente senza ulteriori configurazioni. Una volta referenziata, la libreria sarà a nostra disposizione al di sotto del namespace *CarlosAg.ExcelXmlWriter*, namespace che è quindi opportuno inserire tra i nostri using affinché non sia necessario scriverlo ogni volta:

```
using CarlosAg.ExcelXmlWriter;
```

Oltre a questa modalità è possibile anche installare la DLL nella *GAC* in modo tale da poterla utilizzare dall'interno di qualsiasi applicazione eseguita sulla macchina locale. In tal caso è necessario eseguire il download della versione *Strongly Named* della DLL che lo stesso autore ci mette a disposizione sul suo sito e registrare questa DLL nella GAC impartendo al prompt dei comandi di Windows il comando:

```
gacutil /i <path della DLL>
```

Oppure ancora più semplicemente digitando “assembly” (escluso le virgolette) al prompt dei comandi per aprire la *GAC* e trascinare con il mouse la DLL all'interno di questa finestra.
Per quanto riguarda il *Code Generator*, anche nel suo caso non è necessaria alcuna installazione. È sufficiente lanciare il file *CarlosAg.ExcelXmlWriter.Generator.exe* per avviare l'applicativo e poterlo utilizzare immediatamente.
Vediamo quindi come utilizzare *Excel Xml Writer Library* per creare un documento Excel che contenga dodici fogli di lavoro, uno per ogni mese dell'anno, con all'interno di essi un elenco di voci di spesa per ciascun giorno. Avremo nella prima cella la data del giorno, nella seconda l'importo della spesa mentre la terza cella conterrà una descrizione della spesa. Per semplicità genereremo automaticamente tutti i fogli di lavoro e le righe, inserendo in essi dei valori casuali ma è ovvio che in un caso reale questi valori potranno provenire da database, da un file XML o da qualsiasi altra fonte se non direttamente dalla inter-

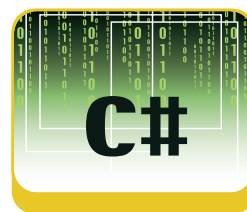
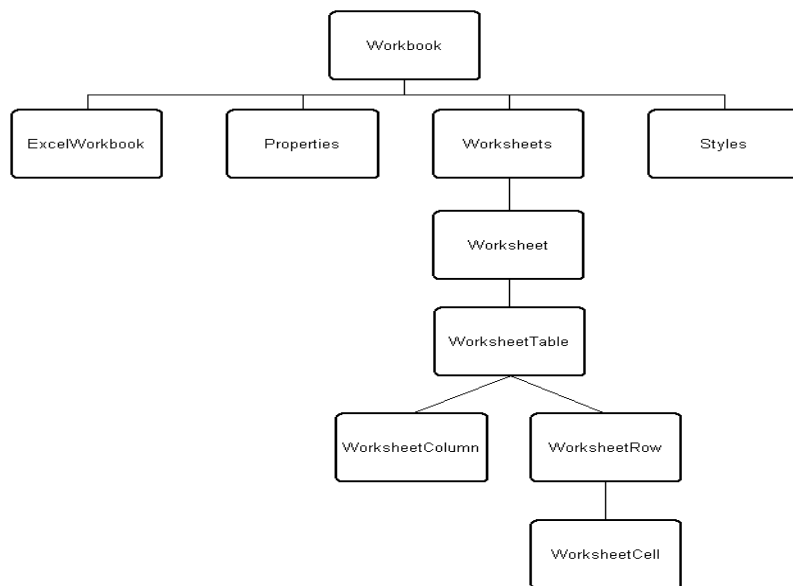



Figura 1: Gerarchia delle classi principali della libreria

SPECIFICHE DEL FORMATO XML SPREADSHEET

Tutti coloro che fossero interessati a comprendere meglio il formato XML Spreadsheet di Excel, possono trovare le specifiche complete di questo formato all'indirizzo: http://msdn2.microsoft.com/en-us/library/aa140066(office.10).aspx

Mentre qui: http://office.microsoft.com/en-us/excel/HA010346391033.aspx sono elencate le limitazioni dell'Xml Spreadsheet rispetto al formato originale di Excel.

faccia utente. Inoltre l'ultima riga di ciascun foglio conterrà anche una cella con il totale degli importi, così da poter vedere anche l'inserimento di formule. Ecco come si presenta la gerarchia delle classi principali di Excel Xml Writer Library:

IMPLEMENTAZIONE

Avviamo Visual Studio e creiamo un nuovo sito Web da *File -> New Web Site...*, scegliamo quindi il template *ASP.NET Web Site*, assicuriamoci che come *Language* ci sia Visual C# e premiamo OK. Apriamo quindi la pagina *Default.aspx* ed inseriamo in essa una *TextBox* ed un *Button*. La TextBox ci servirà per specificare il nome del file Excel che andremo a generare. Il passo successivo consiste nel referenziare la DLL *CarlosAg.ExcelXmlWriter.dll* che troveremo nella cartella in cui avremo preventivamente scompattato il file zip recuperato in precedenza. Fatto tutto questo, possiamo quindi fare doppio clic con il mouse sul Button appena aggiunto per inserire il codice necessario alla generazione del documento Excel.
Per chiarezza e pulizia del codice, manteniamo separati i diversi momenti relativi alla generazione del file Excel e all'invio al client del file stesso. Per questo motivo creiamo cinque distinti metodi ognuno dedicato ad un particolare compito. Partiamo con il codice da inserire nel gestore dell'evento *Click* del Button, il quale funge da entry point della nostra applicazione di esempio:

```
if (txtFilename.Text != "")
{
    // Creiamo un nuovo documento Excel
    Workbook book = ();

    // Impostiamo le proprietà generiche del
                                    documento
    GenerateProperties(book);

    // Creiamo gli stili da applicare alle celle
    GenerateStyles(book);

    int monthNumber = 0;
    int currentYear = .Now.Year;

    foreach (month
        .CurrentCulture.DateTimeFormat.MonthNames)
    {
      // Verifichiamo che month sia diverso da
            stringa vuota perchè l'array MonthNames
      // contiene 13 posizioni in quanto in alcune
                culture esiste un tredicesimo mese
      if (month != "")
      {
        monthNumber++;

        // Creiamo un nuovo foglio di lavoro
        Worksheet sheet =
                      book.Worksheets.Add(month);

        // Giorni nel mese
        int daysInMonth = .DaysInMonth(currentYear,
                                    monthNumber);

        // Creiamo l'header del foglio di lavoro
        GenerateHeader(sheet);

        // Generiamo una riga per ogni giorno del
                                            mese
        GenerateSheet(sheet, currentYear,
                      monthNumber, daysInMonth);

        // Chiudiamo il foglio con la riga del totale
        GenerateFooter(sheet, daysInMonth);
      }
    }

    // Inviamo al browser il documento generato
    ShowWorkbook( book, txtFilename.Text);
}
```

Abbiamo creato un nuovo oggetto di tipo *Workbook* che rappresenta l'intero documento Excel. Quindi abbiamo impostato le proprietà del documento e gli stili in esso presenti che dovranno poi essere applicati alle celle in base al loro contenuto. Ciclando la proprietà *MonthNames* abbiamo recuperato tutti i nomi dei mesi e li abbiamo utilizzati per creare altrettanti fogli di lavoro. Infine con *ShowWorkbook* inviamo tutto al client per essere salvato o visualizzato nel browser.
Vediamo il contenuto dei singoli metodi:

Metodo GenerateProperties

Con questo metodo impostiamo le proprietà dell'intero documento Excel:

```
private void GenerateProperties(Workbook book)
{
  // Indice del foglio di lavoro da visualizzare all'avvio
  book.ExcelWorkbook.ActiveSheetIndex = 0;

  // Autore
  book.Properties.Author = "Gianni";

  // Titolo documento
  book.Properties.Title = "Le spese del 2007";

  // Data di creazione
  book.Properties.Created = DateTime.Now;
}
```

Metodo

Il metodo *GENERATESTYLESHEET* crea invece gli stili:

```
private void GenerateStyles(Workbook book)
 {
   // Aggiungiamo uno stile generico al documento
   WorksheetStyle style =
                        book.Styles.Add("HeaderStyle");

   style.Font.FontName = "Courier";
   style.Font.Size = 14;
   style.Font.Bold = true;
   style.Alignment.Horizontal =
                       StyleHorizontalAlignment.Center;
   style.Font.Color = "Black";
   style.Interior.Color = "LightGray";
   style.Interior.Pattern = StyleInteriorPattern.Solid;

   // Uno stile di default per gli altri elementi del
                                            documento
   style = book.Styles.Add("Default");
   style.Font.FontName = "Courier";
   style.Font.Size = 10;

   // Uno stile per le date
   style = book.Styles.Add("ShortDateStyle");
   style.Font.FontName = "Courier";
   style.NumberFormat = "Short Date";
   style.Alignment.Horizontal =
                       StyleHorizontalAlignment.Center;

   // Uno stile per i valori che eccedono un certo
                                               limite
   style = book.Styles.Add("WarningStyle");
   style.Font.FontName = "Courier";
   style.NumberFormat = "#,##0.00";
   style.Font.Color = "#ee0000";

   // Ed uno stile per i valori monetari
   style = book.Styles.Add("CurrencyStyle");
   style.Font.FontName = "Courier";
   style.NumberFormat = "#,##0.00";
 }
```

In questo modo abbiamo registrato nel documento Excel tutti gli stili che poi potremo successivamente applicare alle celle. È evidente quindi che uno stile per poter essere utilizzato deve innanzi tutto essere definito a livello di *Workbook*.

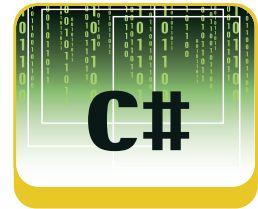
Metodo GenerateHeader

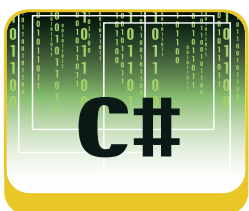
Passiamo quindi al metodo *GenerateHeader* che crea le righe di intestazione di ogni foglio di lavoro:

```
private void GenerateHeader(Worksheet sheet)
{
   // Aggiungiamo tre colonne di dimensione
                                          specifica
   sheet.Table.Columns.Add(new
                              WorksheetColumn(80));
   sheet.Table.Columns.Add(new
                             WorksheetColumn(110));
   sheet.Table.Columns.Add(new
                             WorksheetColumn(300));

   // Aggiungiamo una nuova riga
   WorksheetRow row = sheet.Table.Rows.Add();

   // Definiamo due celle della nuova riga dandogli
                                        un contenuto
   // ed applicando ad esse lo stile prima creato
   row.Cells.Add(new WorksheetCell("Data",
                                     "HeaderStyle"));
   row.Cells.Add(new WorksheetCell("Importo
                            (euro)", "HeaderStyle"));

   // Definiamo una terza cella applicando ad essa un
                           merge di due celle contigue
   WorksheetCell cell = row.Cells.Add("Note");
   cell.MergeAcross = 1;
   cell.StyleID = "HeaderStyle";

   // Inseriamo una riga vuota
   row = sheet.Table.Rows.Add();
}
```

Metodo GenerateSheet

Siamo arrivati quindi al metodo principale dell'applicazione che crea le righe contenenti i valori:

```
private void GenerateSheet(Worksheet sheet, int
       currentYear, int monthNumber, int daysInMonth)
{
   for (int i = 1; i <= daysInMonth; i++)
   {
     // Creazione nuova riga
```

ALCUNE NOTE SU CARLOS AGUILAR MARES

Carlos Aguilar Mares, come da lui stesso riportato nella sezione About me del suo sito (http://www.carlosag.net/aboutme.aspx), è uno sviluppatore che lavora in Microsoft e che dopo aver lavorato per circa due anni nel team di sviluppo di ASP.NET è recentemente passato nel team di Internet Information Server 7. Oltre al sito www.carlosag.net in cui pubblica i suoi lavori più interessanti, ha anche un blog ufficiale sul sito MSDN raggiungibile all'indirizzo: http://blogs.msdn.com/CarlosAg/

```
WorksheetRow row = sheet.Table.Rows.Add();

// Scriviamo una data nella prima cella
row.Cells.Add(new DateTime(currentYear,
        monthNumber, i).ToShortDateString(),
                DataType.String,
                "ShortDateStyle");
// Inseriamo un importo random nella seconda
                                        cella
Random rnd = new Random(i * monthNumber);
decimal importo =
        (decimal)Math.Round(rnd.Next(100) /
                (rnd.NextDouble()*10), 2);
WorksheetCell cell =
row.Cells.Add(importo.ToString().Replace(",", "."),
                DataType.Number,
                "CurrencyStyle");

// Se il valore supera i 50 euro, la cella viene
                                evidenziata
// applicando uno stile opportuno
if (importo > 50)
{
  cell.StyleID = "WarningStyle";
}
// Nota nella terza cella
cell = row.Cells.Add("nota");
cell.MergeAcross = 1;
}
}
```

In questo metodo per ogni giorno del mese considerato, viene generata una riga avente nella prima cella la data del giorno, nella seconda un importo calcolato in maniera casuale, mentre nella terza cella una nota testuale semplice. Notiamo l'enumerato *DataType* che permette di specificare il tipo di dato contenuto nella cella. Questo parametro del metodo *Add* della collection *Cells* è molto importante perché da questo dipende il modo in cui il valore contenuto nella cella sarà visualizzato.
I valori possibili sono:

- _ Boolean
- _ DateTime
- _ Error
- _ Integer
- _ NotSet
- _ Number
- _ String

Per quanto riguarda invece il modo in cui i valori verranno formattati, sarà lo stile applicato alla cella a definirlo attraverso la proprietà *NumberFormat* della classe *WorksheetStyle*.

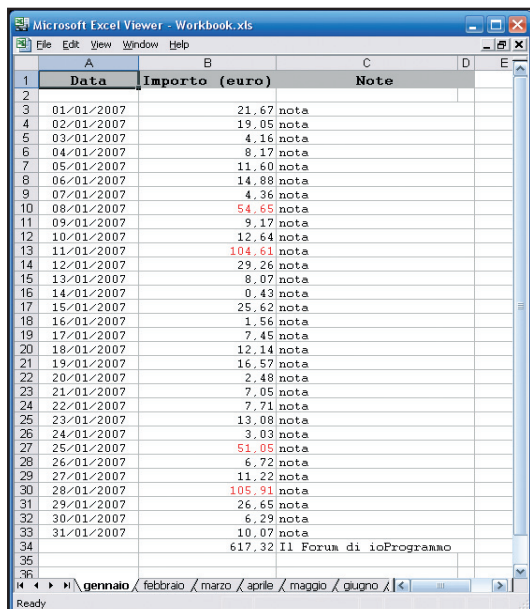
Metodo GenerateFooter

Infine il metodo *GenerateFooter* chiude il foglio di lavoro aggiungendo una riga con il totale dei valori:

```
private void GenerateFooter
                (Worksheet sheet, int rowsNumber)
{
  // Aggiungiamo l'ultima riga
  WorksheetRow row = sheet.Table.Rows.Add();
  // Una cella vuota
  row.Cells.Add();

  // Aggiungiamo la formula per il totale in fondo
                        alla seconda colonna
  WorksheetCell cell = row.Cells.Add();
  cell.Formula = string.Format("=SUM(R[-{0}]C:R[-
                        1]C)", rowsNumber);
  cell.StyleID = "CurrencyStyle";

  // Aggiungiamo un link nell'ultima cella
  cell = row.Cells.Add();
  cell.Data.Text = "Il Forum di ioProgrammo";
  cell.HRef = "http://www.ioProgrammo.it";
  cell.MergeAcross = 1;
}
```

Attraverso la proprietà *Formula* della classe *WorksheetCell*, abbiamo impostato la funzione matematica di Excel, *SUM* che effettua la somma dei valori contenuti nelle celle sovrastanti. Così come è stata usata la funzione *SUM*, allo stesso modo è possibile utilizzare una qualsiasi delle altre funzioni standard di Excel, quali a puro titolo di esempio: *AVERAGE, ROUND, ODD*, e così via. In generale la proprietà *Formula* può contenere qualsiasi espressione che possiamo scrivere all'interno della *Formula Bar* di Excel.
A questo punto il nostro documento Excel è pronto e possiamo quindi inviarlo al browser dell'utente affinché questo possa o visualizzarlo immediatamente oppure salvarlo sul proprio PC. Il metodo *ShowWorkbook* si occupa di questo:

```
private void ShowWorkbook( Workbook workbook,
                        string fileName)
{
  if (fileName != null && fileName != "")
  {
    const string m_Http_Attachment =
                "attachment;filename=";
    const string m_Http_Content =
                "content-disposition";
    HttpResponse m_Response =
                HttpContext.Current.Response;

    if (!fileName.EndsWith(".xls"))
    {
      fileName = fileName + ".xls";
    }
```

```
            m_Response.Clear();
            m_Response.ClearContent();
            m_Response.ClearHeaders();
            m_Response.Buffer = true;

            workbook.Save(m_Response.OutputStream);

            m_Response.AddHeader(m_Http_Content,
  m_Http_Attachment + Path.GetFileName( fileName));
            m_Response.ContentEncoding = Encoding.UTF8;
            m_Response.Charset = "UTF-8";
  m_Response.Cache.SetCacheability
                          (HttpCacheability.NoCache);

            Response.ContentType = "text/xml";
            Response.Flush();
            Response.End();
        }
  }
```

Per salvare il documento Excel creato, la classe *Workbook* della nostra *Excel Xml Writer Library* offre il metodo *Save* che ha due overload. Uno permette di salvare il documento Excel nell'*OutputStream* della *HttpResponse* corrente, mentre l'altro permette di salvare il documento direttamente su file (sul server). Il primo overload è quindi più flessibile in quanto ci permette di salvare il file sulla macchina dell'utente piuttosto che sul server. La scelta naturalmente dipenderà poi dalla specifica applicazione in cui si andrà ad utilizzare questa libreria. Ecco infine il risultato finale:

Microsoft Excel Viewer - Workbook.xls

	A	B	C
1	Data	Importo (euro)	Note
2			
3	01/01/2007	21,67	nota
4	02/01/2007	19,05	nota
5	03/01/2007	4,16	nota
6	04/01/2007	8,17	nota
7	05/01/2007	11,60	nota
8	06/01/2007	14,88	nota
9	07/01/2007	4,36	nota
10	08/01/2007	54,65	nota
11	09/01/2007	9,17	nota
12	10/01/2007	12,64	nota
13	11/01/2007	104,61	nota
14	12/01/2007	29,26	nota
15	13/01/2007	8,07	nota
16	14/01/2007	0,43	nota
17	15/01/2007	25,62	nota
18	16/01/2007	1,56	nota
19	17/01/2007	7,45	nota
20	18/01/2007	12,14	nota
21	19/01/2007	16,57	nota
22	20/01/2007	2,48	nota
23	21/01/2007	7,05	nota
24	22/01/2007	7,71	nota
25	23/01/2007	13,08	nota
26	24/01/2007	3,03	nota
27	25/01/2007	51,05	nota
28	26/01/2007	6,72	nota
29	27/01/2007	11,22	nota
30	28/01/2007	105,91	nota
31	29/01/2007	26,65	nota
32	30/01/2007	6,29	nota
33	31/01/2007	10,07	nota
34		617,32	Il Forum di ioProgrammo
35			
36			

gennaio / febbraio / marzo / aprile / maggio / giugno

Ready

Figura 2: Il documento Excel generato

IL CODE GENERATOR

Unitamente alla *Excel Xml Writer Library*, è quasi indispensabile utilizzare il *Code Generator*, un tool che a partire da un file Excel esistente, genera tutto il codice necessario per riprodurre lo stesso file utilizzando la libreria *Excel Xml Writer*. È molto comodo, infatti, creare un template di come vogliamo che appaia il nostro documento Excel (soprattutto per quanto riguarda gli stili) e lasciare poi al *Code Generator* l'onere di generare tutto il codice necessario per ricreare lo stesso documento da programma. Il nostro compito si ridurrà quindi solo alla modifica del codice generato per adattarlo alla nostra applicazione. L'utilizzo di *Code Generator* è davvero un gioco da ragazzi, è sufficiente avviarlo, premere il tasto *Load Workbook* in alto a sinistra, selezionare dal disco il file Excel che avremo preventivamente creato e quindi, dopo l'elaborazione, premere il tasto *Save* per salvare il codice prodotto sotto forma di classe. Potremo inoltre scegliere tra quattro differenti linguaggi per la generazione del codice che sono: C#, VB.NET, J# e Jscript.

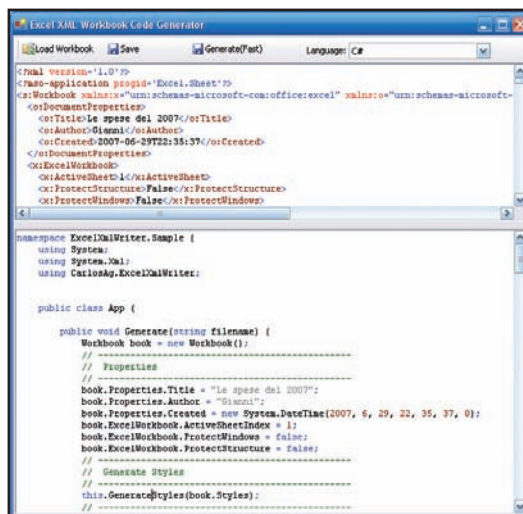
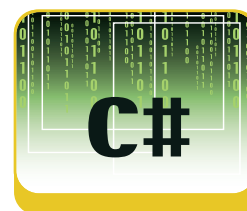



Figura 3: Il Code Generator

CONCLUSIONI

Per questioni di spazio in questo articolo abbiamo visto solo le basi dell'utilizzo della *Excel Xml Writer Library*, ma questa libreria offre naturalmente molto di più. Del tutto gratuitamente possiamo quindi produrre documenti Excel anche di una certa complessità per arricchire le nostre applicazioni senza accollarci l'onere di dover scrivere il codice di generazione del formato *XML Spreadsheet* manualmente. Non possiamo quindi che ringraziare l'autore per questo ottimo prodotto e sperare che come lui altri volenterosi programmatori in giro per il mondo ci forniranno ancora di queste opportunità per migliorare ed agevolare il nostro lavoro di sviluppatori.

Gianni Malanga

L'AUTORE

Lavora da più di 8 anni con tecnologie Microsoft in particolare nel campo dello sviluppo Web. Sviluppa in .NET ormai da alcuni anni e ha svolto diverse docenze per corsi di formazione professionali. Dopo aver lavorato alle dipendenze di importanti realtà locali, attualmente ha intrapreso la libera professione costituendo la ditta individuale Kaone Consulting che si occupa di consulenza informatica per PMI su tutto il territorio nazionale. Gli si può scrivere all'indirizzo kaone@email.it e il suo blog è disponibile su: http://thekaone.blogspot.com

TUTTI IN RETE CON IL FRAMEWORK .NET

HTTP, POP3, SMTP, FTP. SONO ACRONIMI ENTRATI NEGLI ULTIMI ANNI NEL NOSTRO MONDO… MA COSA C'È DIETRO? ANALIZZIAMO I PROTOCOLLI PIÙ COMUNI E MOSTRIAMO COME SFRUTTARLI ALL'INTERNO DELLE NOSTRE APPLICAZIONI

Nel numero scorso abbiamo introdotto i principali protocolli di rete, l'uso dei socket e abbiamo sviluppato un piccolo server Web. In questo numero continueremo la nostra trattazione del networking e paralremo di tre nuovi protocolli: ftp, smtp e pop3.
L'FTP (*File Transfer Protocol*) è un protocollo utilizzato per il trasferimento di dati da un computer ad un altro utilizzando la rete. In particolare, la rete deve supportare il protocollo TCP/IP. Le entità coinvolte in una sessione FTP sono due: un server che utilizza un software per ricevere e processare le richieste FTP ed un client che invia delle richieste al server. Il modello del protocollo FTP è connesso ovvero, una volta stabilita la connessione, il client può eseguire una serie di operazioni sul server prima di effettuare la disconnessione. Nel Framework .Net in versione 1.0 e 1.1 non sono presenti librerie per la gestione delle connessioni FTP. Il Framework 2.0 fornisce le classi **FtpWebRequest** e **FtpWebResponse**. Sono due classi abbastanza comode da utilizzare ma hanno un limite: utilizzano il modello disconnesso delle classi appena viste per la gestione delle richieste HTTP. In pratica, ad ogni richiesta FTP, occorre inviare le credenziali di accesso al server e collegarsi. Nel nostro esempio, realizziamo un client FTP che utilizza il modello disconnesso di .Net per le richieste.

Fig. 1: Le funzioni del nostro client FTP

REQUISITI

Conoscenze richieste
Microsoft.Net

Software
Visual Studio 2005

Impegno

Tempo di realizzazione

Creiamo una classe **FTPManager** per la gestione delle operazioni FTP. Il costruttore è molto semplice: non fa altro che impostare il valore delle proprietà. Tutte le funzioni della classe FTPManager utilizzano un metodo base **Connect** che crea una richiesta FTP al server tramite il metodo *Create* ed associa alla richiesta le credenziali di accesso da inviare al server.

```
''' <summary>
''' Effettua la connessione
''' </summary>
''' <param name="theUri">l'uri a cui
                    connettersi</param>
Private Sub Connect(ByVal theUri As String)
    _Request = CType(FtpWebRequest.Create(New
                    Uri(theUri)), FtpWebRequest)
    _Request.Credentials = New
        NetworkCredential(_UserName, _Password)
    _Request.KeepAlive = False
End Sub
```

Il metodo *Create* restituisce un oggetto di tipo *WebRequest* che convertiamo in *FtpWebRequest*.
Vediamo, adesso, come esempio i metodi **GetFileList** e **Dowload**, usati rispettivamente per ottenere la lista dei file presenti in una cartella del server FTP e per scaricarli in locale.
Otteniamo la lista dei file:

```
''' <summary>
''' Restituisce la lista dei file
''' </summary>
''' <param name="StartDirectory">la cartella da
                    controllare</param>
''' <returns></returns>
Public Function GetFileList(ByVal StartDirectory As
                    String) As List(Of String)

    Dim fileList As List(Of String) = New List(Of
                    String)()

    Dim result As StringBuilder = New
                    StringBuilder()
```

```
        _Uri = "ftp://" & _Server & "/" & StartDirectory

        Connect(_Uri)
        _Request.Method =
                    WebRequestMethods.Ftp.ListDirectory

        Dim response As WebResponse =
                                _Request.GetResponse()
        Dim reader As StreamReader = New
         StreamReader(response.GetResponseStream())

        Try
            Dim currentLine As String = ""
            While (True)
                currentLine = reader.ReadLine()
                If (currentLine <> Nothing) Then
                    fileList.Add(currentLine)
                Else
                    Exit While
                End If
            End While

        Finally
            reader.Close()
            response.Close()
        End Try

        Return fileList
    End Function
```

Il metodo restituisce una lista di stringhe in cui saranno presenti i nomi dei file presenti della cartella scelta. Come prima cosa, costruiamo l'URI a cui inviare la richiesta:

```
_Uri = "ftp://" & _Server & "/" & StartDirectory
```

Eseguiamo il metodo *Connect* - visto prima - passandogli come parametro la variabile _Uri. Impostiamo, quindi, il metodo della richiesta a *ListDirectory* in modo da ottenere in risposta la lista delle cartelle.

```
_Request.Method =
                WebRequestMethods.Ftp.ListDirectory
```

Non ci resta che leggere la risposta:

```
Dim response As WebResponse =
                            _Request.GetResponse()
Dim reader As StreamReader = New
        StreamReader(response.GetResponseStream())
```

Il metodo **GetResponse** della classe *WebRequest* for-

nisce un oggetto di tipo *WebResponse*; a questo punto è sufficiente ottenere uno *StreamReader* tramite il metodo **GetResponseStream** e leggerlo tramite un semplice ciclo *while*.
Effettuiamo, adesso, il download di un singolo file.

```
''' <summary>
''' Scarica un file
''' </summary>
''' <param name="remoteDirectory">directory remota</param>
''' <param name="fileName">nome file</param>
''' <param name="outputDirectory">directory locale</param>
Public Sub Download(ByVal remoteDirectory As String, ByVal fileName As String, ByVal outputDirectory As String)

    Dim downloadStream As FileStream = New FileStream(outputDirectory & "\\" & fileName, FileMode.Create)

    If (remoteDirectory <> "") Then
        _Uri = "ftp://" & _Server & "/" & remoteDirectory & "/" & fileName
    Else
        _Uri = "ftp://" & _Server & "/" & fileName
    End If

    Connect(_Uri)
    _Request.Method = WebRequestMethods.Ftp.DownloadFile

    Dim response As FtpWebResponse = CType(_Request.GetResponse(), FtpWebResponse)
    Dim responseStream As Stream = response.GetResponseStream()

    Dim fileSize As Long = response.ContentLength
    Dim bufferSize As Integer = 2048
    Dim bytesRead As Integer

    Dim buffer(bufferSize) As Byte

    Try
        While (True)
            bytesRead = responseStream.Read(buffer, 0, bufferSize)
            If (bytesRead > 0) Then
                downloadStream.Write(buffer, 0, bytesRead)
            Else
                Return
            End If

        End While
    Finally
        responseStream.Close()
        downloadStream.Close()
        response.Close()
    End Try
End Sub
```

Creiamo un *FileStream* in cui andremo a salvare il file scaricato del server FTP:

```
Dim downloadStream As FileStream = New
    FileStream(outputDirectory & "\\" & fileName, FileMode.Create)
```

Impostiamo, quindi, l'URI ed eseguiamo il metodo Connect.
Il metodo della richiesta dovrà essere impostato a **DownloadFile**.

```
_Request.Method =
    WebRequestMethods.Ftp.DownloadFile
```

Una volta ottenuto lo Stream di risposta, ne leggiamo il contenuto e salviamo i Bytes letti nel *FileStream* creato in precedenza.
Nel CD allegato sono presenti i metodi per effettuare altre operazioni comuni:

- **Upload**: esegue l'upload di un file sul server.
- **Delete**: cancella un file.
- **GetFileDetails**: restituisce la lista dei file di una cartella, con i dettagli per ciascun file.
- **GetFileSize**: recupera la dimensione di un file.
- **Rename**: rinomina un file.
- **CreateDirectory**: crea una directory.

Vediamo, quindi, un esempio di download.
Nella Form abbiamo tre TextBox:

- txt_localSaveDirectory: la directory su cui salvare il file
- txt_remoteDownloadDirectory: la directory sul server
- txt_remoteDownloadFile: il file da scaricare

Il pulsante di download esegue il seguente codice:

```
Try
    Dim ftp As FTPManager = New FTPManager(txt_ftpAddress.Text, txt_username.Text, txt_password.Text)
ftp.Download(txt_remoteDownloadDirectory.Text, txt_remoteDownloadFile.Text, txt_localSaveDirectory.Text)
    MessageBox.Show("Download dompletato")
Catch ex As Exception
    MessageBox.Show("Errore: " & ex.Message)
End Try
```

Viene istanziato un oggetto di tipo FTPManager ed

eseguito il metodo Download.
Sicuro punto a favore delle classi fornite dal Framework è la loro semplicità d'uso. Di contro, il modello disconnesso non è il massimo in fatto di performance e sicurezza.
Pensiamo di dover fare l'upload di 100 file su una cartella: dovremo effettuare 100 connessioni e 100 disconnessioni invece di una soltanto.
È comunque possibile implementare dei client FTP più complessi utilizzando i Socket per creare una connessione permanente ed inviare al server FTP i singoli comandi.

LA POSTA ELETTRONICA

Il protocollo POP (*Post Office Protocol*) è un protocollo di livello 5 nella pila TCP/IP, nato per garantire l'accesso ad account di posta elettronica tramite user e password.
Il server funziona implementando un Listener TCP che resta, in genere, in attesa sulla porta 110.
I client comunicano con il server inviandogli dei comandi:

- **USER Nuome_Utente**: invia la username per la connessione.
- **PASS Password_Utente**: invia la password.
- **LIST**: legge la lista dei messaggi su server (restituisce id e dimensione).
- **RETR Id_Messaggio**: restituisce il contenuto del messaggio.
- **DELE Id_Messaggio**: cancella un messaggio.
- **QUIT**: esce.

Per implementare il nostro client di posta elettronica, non dobbiamo far altro che scrivere un programma capace di instaurare una connessione con un server POP, inviargli dei semplici comandi e leggere le risposte!

Definiamo, come prima cosa, la struttura di un messaggio POP3 tramite la classe **Pop3Message**. Questa ha delle proprietà che definiscono il numero del messaggio, il mittente, la data di ricezione etc.

```
Private _MessageNumber As Long 'Numero
                                   messaggio
Private _MessageSize As Long 'Dimensione
                                   messaggio
Private _DateReceived As String 'Data di ricezione
Private _From As String 'Mittente
Private _FromName As String 'Nome mittente
Private _Subject As String 'Oggetto
```

IL CODICE ALLEGATO

Gli esempi nell'articolo sono in Visual Basic; per chi fosse interessato, nel CD allegato è presente tutto il codice sorgente anche in C#.

```
Private _Message As String 'Messaggio
```

Il costruttore non fa altro che valorizzare le proprietà del messaggio.
Abbiamo poi dei metodi statici:

- **GetFrom**: restituisce il mittente.
- **GetFromName**: restituisce il nome del mittente.
- **GetDate**: restituisce la data.
- **GetSubject**: restituisce l'oggetto.
- **GetBody**: restituisce il corpo.

I metodi sono molto semplici ed il codice è presente nel CD allegato.
La classe che si occupa della comunicazione con il server di posta è la **Pop3Client**.
Come possiamo vedere dalla definizione, essa eredita da **TcpClient**.

```
Public Class Pop3Client
    Inherits TcpClient
```

Definiamo, quindi, delle proprietà per gestire la connessione, inviare messaggi al server e leggerne le risposte.

```
Private _Server As String 'Il server Pop
Private _Username As String 'nome utente
Private _Password As String 'password
Private _Port As Integer 'la porta
Private _ServerMessage As String 'un messaggio
                                    inviato al server
Private _ServerResponse As String 'la risposta
                                    ottenuta dal server
```

Vediamo, adesso, i metodi per la gestione della casella di posta:

- **Connect**: effettua la connessione.
- **Disconnect**: effettua la disconnessione.
- **ListMessages**: restituisce la lista messaggi sul server.
- **ReadMessage**: legge un singolo messaggio.
- **DeleteMessage**: cancella un messaggio.

Tutti i metodi della classe utilizzano le seguenti funzioni per comunicare con il server di posta:

- **SendServerMessage**: invia un messaggio al server.
- **ReadServerResponse**: riceve la risposta.

Analizziamole nel dettaglio.
Il metodo **SendServerMessage** invia dei messaggi contenenti dei comandi al server:

```
    ''' <summary>
    ''' Invia un messaggio al server
    ''' </summary>
    ''' <param name="message"></param>
    Private Sub SendServerMessage(ByVal message
                                        As String)

        Dim enc As ASCIIEncoding = New
                                        ASCIIEncoding()
        Dim buffer(1024) As Byte
        buffer = enc.GetBytes(message)
        Dim sendStream As NetworkStream =
                                        GetStream()
        sendStream.Write(buffer, 0, buffer.Length)

    End Sub
```

Riceve in ingresso un messaggio in forma di stringa, lo codifica in un array di Byte con il metodo **GetBytes** e lo invia tramite un *NetworkStream* al server. Lo Stream viene ottenuto utilizzando il metodo **GetStream** della classe *System.Net.Sockets.TcpClient* da cui la nostra classe eredita.
Il metodo **ReadServerResponse** legge i messaggi inviati dal server in risposta alle richieste:

```
    ''' <summary>
    ''' Riceve la risposta
    ''' </summary>
    ''' <returns></returns>
    Private Function ReadServerResponse() As String

        Dim enc As ASCIIEncoding = New
                                        ASCIIEncoding()
        Dim responseBuffer(1024) As Byte
        Dim stream As NetworkStream = GetStream()
        Dim count As Integer = 0
        While (True)

            Dim innerBuffer(2) As Byte
            Dim bytesRead As Integer =
                            stream.Read(innerBuffer, 0, 1)
            If (bytesRead = 1) Then
                responseBuffer(count) = innerBuffer(0)
                count = count + 1
                If (innerBuffer(0) = 10) Then
                    Exit While
                End If
            Else
                Exit While
            End If
        End While
        Return enc.GetString(responseBuffer, 0, count)

    End Function
```

Ottiene dal server uno *Stream* tramite il metodo **GetStream,** utilizza, quindi, un ciclo *while* per leggere lo *Stream* (con il metodo *Read* della classe *Stream*) e restituire una stringa contenente la risposta.

NOTE

TCPCLIENT

La classe TCPClient fornisce dei metodi per creare un client TCP. Per funzionare ha bisogno di avere un server TCP in ascolto per collegarsi al quale può utilizzare il metodo Connect. Fornisce inoltre il metodo GetStream per ottenere un NetworkStream da cui poter leggere i dati ricevuti dal server. Ulteriori informazioni sono reperibili all'indirizzo http://msdn2.microsoft.com/it-it/library/system.net.sockets.tcpclient(vs.80).aspx

I due metodi visti sopra sono la base della nostra classe in quanto vengono utilizzati per comunicare con il server di posta.
Vediamo, adesso, come fare per recuperare la lista dei messaggi presenti in una casella di posta tramite la funzione **ListMessages**:

```
''' <summary>
''' Lista messaggi sul server
''' </summary>
''' <returns>i messaggi</returns>
Public Function ListMessages() As List(Of
                                    Pop3Message)

    Dim messageList As List(Of Pop3Message) =
                        New List(Of Pop3Message)()

    _ServerMessage = "LIST" & vbCrLf
    SendServerMessage(_ServerMessage)
    _ServerResponse = ReadServerResponse()

    If (_ServerResponse.Substring(0, 3) <>
                                    "+OK") Then
        Throw New Exception("Unable to read mail")
    End If

    'leggo i messaggi
    While (True)
        _ServerResponse = ReadServerResponse()
        If (_ServerResponse = "." & vbCrLf) Then
            Exit While
        Else
            Dim msg As Pop3Message = New
                                    Pop3Message()
            Dim msgParts() As String =
    _ServerResponse.Replace(vbCrLf, "").Split(" ")
            msg.MessageNumber =
                        Convert.ToInt32(msgParts(0))
            msg.MessageSize =
                        Convert.ToInt32(msgParts(1))
            messageList.Add(msg)
            Continue While
        End If

    End While
    'aggiungo il messaggio completo alla lista
    For Each msg As Pop3Message In messageList
        Dim tmp As Pop3Message =
                ReadMessage(msg.MessageNumber,
                                    msg.MessageSize)
        msg.Message = tmp.Message
        msg.Subject = tmp.Subject
        msg.DateReceived = tmp.DateReceived
        msg.From = tmp.From
```

STREAM
Uno Stream rappresenta un flusso di dati in Input/output come una sequenza di Byte.
La classe Stream del Framework .Net è una classe astratta alla base di tutti i tipi di Stream.
Questi possono essere ad esempio FileStream (per la lettura e scrittura di file) o NetworkStream (per l'invio e la ricezione di dati mediante socket).

```
            End If
        End Try
Catch ex As Exception
        MessageBox.Show("Errore" & ex.Message)
End Try
```

La form completa è presente nel CD allegato e contiene anche i metodi per visualizzare il singolo messaggio o cancellare un messaggio dal server.

RICEVERE LE EMAIL

Il protocollo SMTP, anch'esso di livello 5, consente di inviare dei messaggi e-mail ad un host. Il protocollo utilizza la porta 25 per l'invio dei messaggi.
Per l'invio delle mail, il Framework ci viene molto in aiuto fornendo il namespace **System.Net.Mail** che contiene già tutte le classi ed i metodi che ci servono. La nostra implementazione della classe **MailSender** risulta molto semplice.

```
Imports System
Imports System.Net.Mail
''' <summary>
''' Invio Mail
''' </summary>
Public Class MailSender
    Private mailFrom As String
    Private mailTo As String
    Private smptserver As String
    ''' <summary>
    ''' Costruttore
    ''' </summary>
    ''' <param name="mailFrom">From</param>
    ''' <param name="mailTo">To</param>
    ''' <param name="smptserver">Server
                                        SMTP</param>
    Public Sub New(ByVal mailFrom As String, ByVal
          mailTo As String, ByVal smptserver As String)
        mailFrom = mailFrom
        mailTo = mailTo
        smptserver = smptserver
    End Sub
    Public Sub SendMessage(ByVal subject As String,
     ByVal message As String, ByVal attachments() As
                                             String)
        Dim mailMsg As MailMessage = New
                                        MailMessage()
        mailMsg.Priority = MailPriority.Normal
        mailMsg.From = New MailAddress(mailFrom)
        mailMsg.ReplyTo = New MailAddress(mailFrom)
        mailMsg.To.Add(New MailAddress(mailTo))
        mailMsg.Bcc.Add(New MailAddress(mailFrom))
        mailMsg.Subject = subject
        mailMsg.Body = message

        If (attachments.Length > 0) Then
            For Each attach As String In attachments
                mailMsg.Attachments.Add(New
                                Attachment(attach))
            Next
        End If
        mailMsg.Headers.Add("Return-Path", mailFrom)
        mailMsg.Headers.Add("Date",
                            DateTime.Now.ToString())
        mailMsg.Headers.Add("MIME-Version", "1.0")

        mailMsg.IsBodyHtml = False

        Dim smtpClient As SmtpClient = New
                            SmtpClient(smptserver)
        smtpClient.Send(mailMsg)
    End Sub

End Class
```

Il metodo **SendMessage** non fa altro che creare un oggetto di tipo *MailMessage*, impostarne le proprietà e poi inviarlo.

Vediamo alcune proprietà del messaggio.

- **Priority**: priorità del messaggio.
- **From**: mittente (di tipo MailAddress).
- **ReplyTo**: indirizzo di risposta (di tipo MailAddress).
- **To**: lista destinatari (di tipo MailAddress).
- **Bcc**: lista destinatari in copia nascosta (di tipo MailAddress).
- **Subject**: oggetto.
- **Body**: corpo.
- **Attachments**: la lista di allegati (di tipo Attachment).
- **Headers**: una lista di headers inviati con il messaggio (coppie chiave-valore).
- **IsBodyHtml**: se la mail è in formato HTML.

L'invio viene eseguito istanziando un oggetto di tipo *SmtpClient* ed invocandone il metodo *Send*.

CONCLUSIONI

Come abbiamo potuto notare nel corso dell'articolo, il Framework fornisce una notevole quantità di classi per la gestione dei protocolli di rete e la realizzazione di applicazioni che li utilizzano.
Alcune di esse sono molto semplici, altre richiedono una maggiore conoscenza delle tecniche di programmazione. Sono, comunque, classi molto flessibili e, con un po' di fantasia da parte nostra, riescono a farci gestire qualsiasi aspetto della programmazione di rete.

Carmelo Scuderi

L'AUTORE

Carmelo Scuderi è ingegnere informatico. Si occupa di sviluppo software in ambiente .Net per una società di informatica di Milano. Gestisce un sito ricco di script e manuali per chi si affaccia al mondo della programmazione web (www.morpheusweb.it).

AJAX FACILE FACILE? FALLO CON THINWIRE

ALLA SCOPERTA DI UN FRAMEWORK AJAX OPEN SOURCE (LGPL) INTERAMENTE SCRITTO IN JAVA, CON UNA CURVA DI APPRENDIMENTO VERAMENTE BASSA ED UNO STILE MOLTO SIMILE A QUELLO DI SWING O AWT. IL WEB 2.0 DIVENTA SEMPLICE!

L'avvento e l'utilizzo della tecnologia Ajax ha rivoluzionato sia il modo di concepire il web sia il mondo degli sviluppatori; è stato così determinante da aver partecipato in modo significativo al concepimento del web 2.0. Molta entropia è stata aggiunta tra sviluppatori e progettisti di applicazioni web. Ormai non basta più impiegare framework solidi e ben testati affermatisi nell'era pre-Ajax, impiegare questa tecnologia è diventato un must per ogni programmatore. Al momento vi sono molti framework che supportano in modo più o meno intuitivo la tecnologia Ajax. Molti sono ottime librerie Javascript che wrappano l'oggetto XHR (XmlHttpRequest), altri invece sono basati su Widget grafici i cui comportamenti sono regolati secondo il paradigma Ajax. Questi ultimi mascherano, chi più chi meno, le interazioni tra codice Javascript ed oggetti DOM lato client. Infatti sono capaci di demandare il grosso del lavoro al server e lasciare al browser la sola funzionalità di rendering. In questo articolo analizzeremo le caratteristiche salienti di ThinWire provandolo direttamente sul campo; la versione a cui faremo riferimanto è la 1.2-rc2.

HELLOCLOCK

Iniziamo subito a capire come funziona ThinWire con un esempio banale; successivamente ci cimenteremo nella realizzazione di un'applicazione relativamente complessa. Supponiamo di voler realizzare un *Helloworld* web anzi, meglio, un *HelloClock* in cui cliccando su un pulsante verranno mostratate data e ora corrente.

```
public class HelloClock {
  public static void main(String [] ar){
    Frame frame =
                    Application.current().getFrame();
    frame.setLayout(new TableLayout(new
                                          double[][]{
              {200},{40,50}
               },10,10));
        final Label timeLabel=new Label();
        Button but=new Button("Time!");
frame.getChildren().add(timeLabel.setLimit("0,0"));
        frame.getChildren().add(but.setLimit("0,1"));
        frame.setVisible(true);
        but.addActionListener(Button.ACTION_CLICK,
                                   new ActionListener(){
          public void actionPerformed(ActionEvent ev) {
              Date d=new Date();
               timeLabel.setText(d.toString());
           }
        });
    }
}
```

Queste poche righe di codice bastano per farci capire che il *modus operandi* di questo framework è molto simile a quello della maggior parte delle API utilizzate da Java per lo sviluppo di interfacce grafiche. A parte poche sfumature potremmo cambiare la sezione degli import (non mostrata nell'esempio) ed ottenere un'applicazione Swing. L'oggetto *Application.current()* rappresenta l'applicazione web in esecuzione su un determinato browser; quindi per ogni browser vi sarà un oggetto diverso. A partire da tale oggetto è possibile ottenere il Frame che conterrà di volta in volta la pagina web; infatti all'interno di questo oggetto si possono posizionare altri componenti grafici sfruttando le potenzialità del layout utilizzato. Nel nostro esempio abbiamo utilizzato un TableLayout che struttura il Frame in una colonna larga 200 px suddivisa in due righe alte rispettivamente 40 px e 50 px. I componenti come Frame, che ne possono contenere degli altri, implementano l'interfaccia *Container*. Come possiamo notare questa interfaccia espone il metodo *getChildren()* che ci consente di recuperare la lista degli oggetti contenuti. Nel nostro esempio abbiamo aggiunto una Label ed un Button sfruttando il metodo *add*. Ogni componente possiede il metodo *setLimit* che gli per-

mette di posizionarsi all'interno del Container a cui è stato aggiunto; quindi la Label occuperà la posizione "0,0", mentre il Button la posizone "0,1". Per poter gestire l'evento relativo alla pressione del Button aggiungiamo un *ActionListener* rappresentato da un oggetto istanza di una classe anonima. Ogni volta che verrà premuto il pulsante verrà invocato il metodo *actionPerformed* di tale classe; al suo interno si gestisce l'evento impostando la data corrente nella Label utilizzata. Ma come fa una normale classe con tanto di main a diventare un'pplicazione web? Beh in questo i progettisiti di ThinWire sono stati tanto originali quanto eleganti. La magia risiede nel file descriptor web.xml:

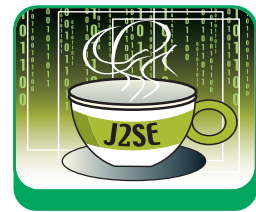
```
<web-app>
  <servlet>
    <servlet-name>thinwire</servlet-name>
    <description>Servlet Engine</description>
    <servlet-class>
        thinwire.render.web.WebServlet
    </servlet-class>
    <init-param>
      <param-name>mainClass</param-name>
     <param-value>
        roby.thinwire.client.impl.HelloClock
     </param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>thinwire</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

In pratica si specifica che tutte le richieste inerenti la webapp in questione sono gestite da un oggetto istanza di WebServlet; questo oggetto delega tale gestione alla classe HelloClock. *L'entrypoint* di tale classe è costituito dal metodo main. Quindi per ogni nuova sessione viene invocato il metodo main, le richieste successive saranno eventualmente gestite dagli oggetti creati all'interno del main che si sono messi in ascolto di determinati eventi.

MESSAGGI PRIVATI

Da qui in poi ci occuperemo di sviluppare un'applicazione volutamente complessa, così facendo potremo capire le possibilità di questo framework e la sua facilità di utilizzo per chi ha qualche esperienza con le GUI di Java. La nostra applicazione web sarà costituita da un servizio di messaggistica privato; tale servizio sarà fruibile previa autenticazione e disporrà delle funzionalità elementari: invio di un nuovo messaggio e consultazione dei messaggi ricevuti ed inviati. Vista la natura didattica dell'articolo e lo spazio a disposizione ci soffermeremo sui punti caldi del progetto; il codice completo è comunque disponibile sul CD allegato.
Iniziamo dal main della nostra applicazione.

```
public class Client {
   private static String parse(String st){
     String res=null;
     res=st.substring(st.indexOf("=")+1);
     return res;
  }
  public static void main(String [] ar){
     WebApp1 wa1=new WebApp1();
     WebApp1.dbURL=parse(ar[0]);
     wa1.buildUI();
  }
}
```

Come possiamo notare viene creato un oggetto istanza di *WebApp1*, viene recuperato un oggetto passato come argomento al metodo main, ed infine viene invocato il metodo buildUI che, come vedremo da qui a poco, ha il compito di costruire le struttura grafica della nostra applicazione. L'argomento passato al metodo main funge da parametro di configurazione per la connessione al DB. Per poter accedere ai parametri di inizializzazione della servlet, attraverso questo

THINWIRE E I LAYOUT

Allo stato attuale ThinWire consente di utilizzare solo due tipi di layout: SplitLayout e TableLayout. Così come per i framework desktop (Swing, AWT, SWT e cosi via) il layout definisce la strategia con cui può essere suddiviso e quindi riempito un Container. Lo SplitLayout serve a suddividere orizzontalmente o verticalmente un Container in due regioni. La peculiarità degli SplitLayout è quella di definire un bordo tra le due regioni che può essere spostato durante l'esecuzione dell'applicazione. Il TableLayout è il layout più potente e quindi anche il più usato nella definizione dei componenti Container (Dialog, Panel e Frame su tutti) di ThinWire. Il suo utilizzo è abbastanza semplice ma va capito bene onde evitare comportamenti inaspettati. Prendiamo in esame un esempio di utilizzo:

```
a.setLayout(new TableLayout(new dou
                           ble[][]{
      {.30,.40,.30},
      {.20,.80}},5,10));
 a.getChildren().add(b.setLi-
 mit("0,1"));
 a.getChildren().add(c.setLi
                          mit("0,0,2,1"));
```

In questo semplice esempio abbiamo munito il Container di un TableLayout; il quale definisce una griglia di tre colonne e due righe. Le colonne hanno le seguenti larghezze percentuali 30%, 40% e 30%. Le due righe invece hanno altezze percentuali pari al 20% e all'80%. Gli ultimi due interi rappresentano la spaziatura dal bordo del Container e la distanza tra le celle della griglia. In questo esempio gli oggetti b e c vengono posizionati il primo in poszione 0,1 (colonna,riga) della griglia, il secondo in posizione 0,0; però il componente c occuperà due celle in orizzontale ed una in verticale.

semplice meccanismo, è necessario definire nel file web.xml oltre al parametro di configurazione

```
<init-param>
   <param-name>dbURL</param-name>
   <param-
  value>jdbc:hsqldb:file:/roby/private/software/inst/
                    tomcat55/webapps/thinw/WEB-
                     INF/data/pma</param-value>
 </init-param>
```

anche un particolare parametro denominato *extraArguments* e valorizzarlo con la stringa *initParam*

```
<init-param>
  <param-name>extraArguments</param-name>
  <param-value>initParam</param-value>
</init-param>
```

Le restanti sezioni del descrittore della servlet sono del tutto simili a quelle presentate nell'esempio precedente.

Fig. 1: La visualizzazione dei messaggi con PMA

QUESTIONE DI DESIGN

È giunto il momento di definire la struttura grafica della nostra applicazione; come si presenterà la nostra applicazione agli utenti? Supponiamo di voler strutturare il layout dell'applicazione nel seguente modo: a sinistra definiamo una colonna in cui sono presenti i pulsanti che ci consentono di eseguire le funzionalità principali, mentre a destra avremo un header in cui posizioneremo il nome dell'applicazione ed un paio di immagini.
La rimanente zona sulla destra della pagina conterrà la parte *core* della nostra applicazione e cambierà di volta in volta a seconda della funzionalità eseguita. Non ci rimane adesso che analizare adesso la classe WebApp1 che definisce tale struttura:

```
public class WebApp1 {
   private UIFactory factory=null;
   public static Application.Local<HashMap>
          session=new Application.Local<HashMap>();
   public static String dbURL=null;
      . . .
}
```

Prima di inoltrarci nell'analisi dei vari metodi notiamo subito che esiste un oggetto factory con visibilità privata istanza di UIFactory, tale oggetto servirà come factory di tutti i widget grafici della nostra applicazione. E' stato definito anche un oggetto statico session, il suo tipo è definito con l'ausilio dei generics; infatti la classe *Application.Local*<...> consente di definire una particolare classe, nel nostro caso abbiamo preferito HashMap, la cui istanza è differente da sessione a sessione. Questo passaggio è molto importante da capire, infatti questo meccanismo introdotto da ThinWire consente di definire un solo oggetto per sessione; però se ci attrezziamo con l'oggetto giusto (un HashMap lo è!) riusciamo a superare questa apparente limitazione. Infine la stringa dbURL serve a definire l'url di connessione al DB. Vediamo adesso come funziona il metodo buildUI.

```
public void buildUI(){
  Frame frame = Application.current().getFrame();
  HashMap hm=new HashMap();
  session.set(hm);
  ViewManager vm=new ViewManager();
  session.get().put(Names.VIEW_MANAGER, vm);
  frame.setTitle("Private Message Application PMA");
frame.getStyle().getBackground().setColor
                          (Color.GREENYELLOW);
  frame.setLayout(new TableLayout
            (new double[][] {.10,.90},{50,.99}},0,0));
  this.factory.buildLoginDialog();
frame.getChildren().add(this.factory.buildNorth
                                       Panel());
  frame.getChildren().add(this.buildInBoxPanel());
frame.getChildren().add(this.factory.buildLeftPanel());
  vm.put(Names.VIEW_FRAME, frame);
  frame.setVisible(true);
}
```

Come al solito la prima cosa da fare è recuperare il Frame che conterrà la nostra pagina web, successivamente viene inserita la HashMap nell'oggetto session discusso in precedenza. L'oggetto vm istanza di ViewManager viene anch'esso inserito nell'oggetto session; già da queste poche righe di codice possiamo notare l'utilità e la semplicità della classe Names i cui membri pubblici e statici servono a dare un nome alle entry della HashMap. Successivamente vengono defi-

niti un titolo ed un colore di background per il Frame. E' giunto il momento di suddividere il Frame secondo le specifiche definite in precedenza.

```
frame.setLayout(new TableLayout(new double[][]{
    {.10,.90},{50,.99}},0,0));
```

E' stato definito un TableLayout che suddivide il Frame in due colonne, la prima larga il 10% della larghezza totale e la seconda il restante 90%. Lo stesso Frame è suddiviso in due righe di cui la prima, utilizzata per l'header, alta 50 pixel mentre la seconda alta per il restante spazio. I successivi due 0 definiscono la spazio dal bordo e la distanza in pixel tra le celle della tabella. A questo punto si delega l'oggetto factory per la costruzione di una Dialog per l'autenticazione. Adesso non ci resta che aggiungere tre Panel nelle zone di loro competenza: a sinistra per il pannello che conterrà la pulsantiera dei comandi, in alto per l'intestazione e la parte centrale per la parte *dinamica* della nostra applicazione. Le ulitme due istruzioni riguardano l'inserimento del Frame nel *ViewManager* ed il comando *setVisible(true)* che lo rende visibile all'utente. Concentriamoci un attimo sull'implementazione del metodo *buildInBoxPanel*:

```
private Panel buildInBoxPanel(){
    ViewManager
      vm=(ViewManager)session.get().get(Names.VIEW
                                        _MANAGER);
    Panel p=this.factory.buildInBoxPanel();
    vm.put(Names.VIEW_INBOX_PANEL, p);
    vm.put(Names.VIEW_VISIBLE_PANEL, p);
    return p;
}
```

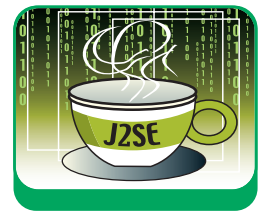
Per prima cosa viene recuperato il ViewManager dall'HashMap contenuta nell'oggetto session; a questo punto si incarica l'oggetto factory di costruire un Panel per la rappresentazione delle mail ricevute (InBox). Tale Panel viene memorizzato nel ViewManager sia come pannello InBox (*Names.VIEW_INBOX_PANEL*) sia come il Panel che in questo momento è visibile al centro della pagina web (*Names.VIEW_VISIBLE_PANEL*). Questo semplice meccanismo ci consentirà di cambiare, in modo molto elementare, il Panel visualizzato a centro pagina.

UIFACTORY

Come abbiamo visto la costruzione vera e propria dell'interfaccia grafica è demendata alla classe UIFactory. Oltre a questa funzionalità si occupa anche di effettuare alcune semplici operazioni sui componenti grafici da essa stessa costruiti. La sua implementazione prevede l'utilizzo del pattern *singleton*.

```
public class UIFactory {
  private static UIFactory obj=null;
  private UIFactory(){}
  public static UIFactory sing(){
    if(obj==null)
        obj=new UIFactory();
    return obj;
  }
  //altri metodi
   . . .
}
```

Forti di questa implementazione possiamo utilizzare la stessa (ed unica) istanza di questa classe in tutti i punti del codice senza molti sforzi. Incominciamo però ad analizzare il metodo che costruisce e definisce il layout della finestra di dialogo utilizzata all'atto del *login* per inserire e verificare le credenziali dell'utente.

```
public Dialog buildLoginDialog(){
  ViewManager
    vm=(ViewManager)WebApp1.session.get().get(Na
                              mes.VIEW_MANAGER);
  Dialog d=new Dialog("Login");
  d.setBounds(20, 20, 430, 200);
  d.setLayout(new TableLayout(new double[][]{
    {.11, .25, .41, .33},
    {.20,.15,.15,.20,.30}},5,5));
  Label userLabel=new Label("username:");
  userLabel.setAlignX(AlignX.RIGHT);
  Label pswLabel=new Label("password:");
  pswLabel.setAlignX(AlignX.RIGHT);
  Label error=new Label("Wrong username or
                                   password");
  error.setAlignX(AlignX.CENTER);
  error.setVisible(false);
  error.getStyle().getFont().setSize(14);
```

NOTA

GLI ESEMPI SUL CD

Sotto la directory ThinWire/src troverete i sorgenti completi delle due applicazioni descritte in questo articolo; sotto la directory ThinWire/webapps troverete invece le due webapp. La webapp thinw relativa all'applicazione PMA necessita della configurazione della stringa di connessione al DB come descritto nell'articolo. Una volta copiate le due webapp sotto la cartella webapps della propria istanza di tomcat, ed avviato lo stesso, è sufficiente visitare gli url http://127.0.0.1:8080/thinw/ e http://127.0.0.1:8080/clock/ per vederle all'opera.

MVC? SÌ GRAZIE

Nella nostra applicazione abbiamo realizzato una versione custom del pattern MVC; infatti la classe ViewManager serve a memorizzare vari widget grafici che possono essere recuperati, manipolati e visualizzati in vari punti della stessa. Sostanzialmente questa classe è composta da un HashMap che indicizza le view in base ad una stringa costante presente nella classe Names. Ad ogni componente grafico, che può generare eventi, è stato associato un listener dedicato; per aderire in modo fedele al pattern MVC tale listener è stato battezzato come Controller. Nel package roby.thinwire.client controller possiamo trovare tutti i controlli dell'applicazione. La parte Model del pattern invece è rappresentata dai DTO e dai servizi di business implementati. Questo approccio può essere riutilizzato abbastanza agevolmente in qualsiasi framework che non supporta nativamente il pattern MVC.

```
error.getStyle().getFont().setColor(Color.RED);
TextField user=new TextField("");
TextField psw=new TextField("");
psw.setInputHidden(true);
```

In questa prima parte del metodo notiamo che viene creata un'istanza della classe Dialog; questa classe definisce graficamente una finestra che si sovrappone alla pagina web. La Dialog in questione, sfruttando il metodo *setBounds*, sarà posizionata alle coordinate 20,20 della pagina sottostante ed avrà lunghezza 430 ed altezza 200. Come al solito è stata munita di un *TableLayout*, in questo caso suddivide la Dialog in quattro colonne secondo la seguente distribuzione percentuale 11%, 25%, 41%, 33%; analogamente suddivide la stessa Dialog in cinque righe seguendo la distribuzione percentuale rappresentata dal secondo array definito all'interno del suo costruttore.
Successivamente vengono costruite tre istanze di *Label*; le prime due si occuperanno di visualizzare la stringhe *username* e *password*. La terza Label sarà visualizzata solo in caso di login errato. E' facile intuire che *setAlignX(AlignX.RIGHT)* serve a posizionare il componente grafico, lungo l'asse X, al centro dello spazio a disposizione. Le ultime tre righe di codice servono alla costruzione di due *TextField*, user è la casella di testo in cui inserire lo username e *psw* dove inserire la password. L'ultima riga di codice serve a nascondere il contenuto del testo inserito nel TextField psw. Soffermiamoci adesso sulla seconda parte del metodo:

```
vm.put(Names.VIEW_DIALOG_USERNAME, user);
vm.put(Names.VIEW_DIALOG_PSW, psw);
vm.put(Names.VIEW_DIALOG_ERROR, error);
Button okB=new Button("Login");
okB.setBounds(10,10,70,20);
LoginButtonController bc=new
LoginButtonController(vm);
okB.addActionListener(Button.ACTION_CLICK,
                                        bc);
d.getChildren().add( userLabel.setLimit("1,1"));
d.getChildren().add(pswLabel.setLimit("1,2"));
d.getChildren().add(user.setLimit("2,1"));
d.getChildren().add(psw.setLimit("2,2"));
d.getChildren().add(okB.setLimit("2,3"));
d.getChildren().add(error.setLimit("0,4,4,1"));
d.setModal(true);
d.setVisible(true);
. . .
vm.put(Names.VIEW_LOGIN_DIALOG, d);
return d;
}
```

Le tre Label discusse in precedenza vengono memorizzate nel ViewManager, dopodiché viene creato un oggetto okB istanza di *Button*; questo *Button* servirà a confermare l'inserimento delle credenziali dell'utente. Ecco comparire il primo *Controller*, si tratta dell'oggetto *bc* istanza della classe *LoginButtonController*. Lo stesso Controller viene aggiunto come ascoltatore dell'evento (action) Button. ACTION_CLICK. Successivamente gli oggetti finora creati vengono inseriti nel Container Dialog attraverso l'invocazione di *getChildren().add*. Bisogna sottolineare che il metodo setLimit serve a fissare la posizione nella griglia in cui l'oggetto viene inserito; la Label *error* in particolare viene inserita nella posizione 0,4 (colonna 0, riga 4), però la sua lunghezza occuperà quattro celle mentre la sua altezza una. Infine la Dialog in questione viene memorizzata nel ViewManager.

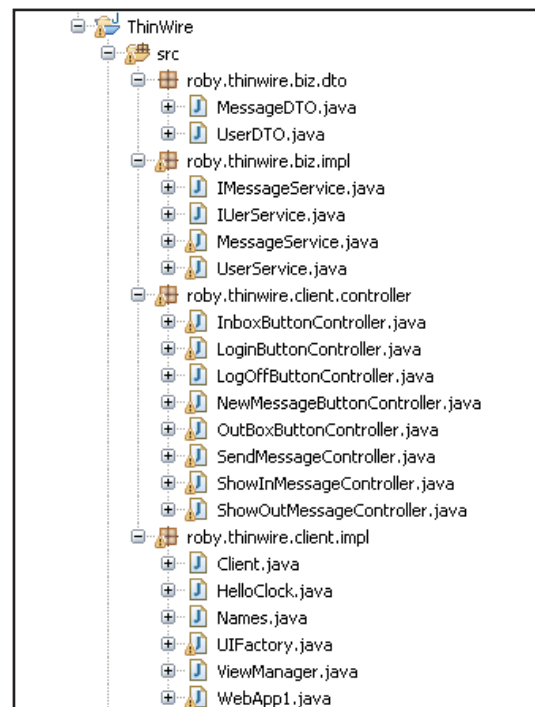



Fig. 2: ***Il progetto visto da Eclipse***

INBOX PANEL

Sempre nella classe UIFactory è presente il metodo buildInBoxPanel, una volta capito il suo funzionamento lo studio del codice relativo agli altri metodi è una pura formalità. In questo Panel si vogliono visualizzare l'elenco di tutti i messaggi ricevuti, e si vuole dare all'utente la possibilità di visualizzare il dettaglio di ogni singolo messaggio.

```
public Panel buildInBoxPanel(){
   Panel p=new Panel();
```

INSTALLARE THINWIRE

Per installare ThinWire è necessario per prima cosa procurarsi lo zip relativo al framework, è possibile effettuarne il download all'indirizzo www.thinwire.com. A questo punto è sufficiente scompattare lo zip e copiare sotto la cartella WEB-INF/lib/ della propria applicazione web i tre jar presenti sotto <thinwire>/demos/helloworld/WEB-INF/lib/

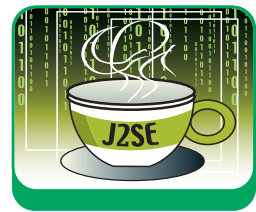
```
p.setLayout(new TableLayout(new double[][]{
  {.15, .70, .15},{.20,.60,.20}
  },30,30        ));
. . .
Label title=new Label("In Box");
title.setLimit("1,0");
```

In questa prima parte del metodo viene creato il Panel, viene fornito di un TableLayout che lo suddivide in tre colonne e tre righe. Successivamente viene creata una Label e viene stabilito che la sua posizione sarà quella relativa alla cella che ha come coordinate la colonna 1 e la riga 0. A questo punto è necessario creare un oggetto istanza di GridBox.

```
GridBox comp = new GridBox();
comp.setVisibleHeader(true);
GridBox.Column col=new GridBox.Column();
col.setName("FROM:");
comp.getColumns().add(col);
col = new GridBox.Column();
col.setName("DATE:");
comp.getColumns().add(col);
col = new GridBox.Column();
col.setName("SUBJECT:");
comp.getColumns().add(col);
col = new GridBox.Column();
col.setName("ID");
comp.getColumns().add(col);
col.setVisible(false);
```

L'oggetto in questione si presenta come una tabella munita di un header dove sono visualizzate le etichette relative alle colonne. Infatti il metodo setVisibleHeader(true) serve ad implementare tale funzionalità.
Successivamente vengono creati tre oggetti ognuno dei quali è una istanza della classe GridBox.Column; in particolare il metodo *setName* serve a definire l'etichetta della colonna in questione. E' utile osservare che l'ultima colonna viene inizializzata e munita di un nome (ID) ma non viene visualizzata; infatti essa sarà utilizzata per referenziare in modo univoco le varie righe dell'oggetto GridBox.

```
ViewManager vm=(ViewManager)
                    WebApp1.session.get().get
                    (Names.VIEW_MANAGER);
vm.put(Names.VIEW_GRIDBOX_INBOX, comp);
comp.addActionListener(GridBox.ACTION_DOUBLE_
       CLICK, new ShowInMessageController(vm));
p.getChildren().add(comp.setLimit("1,1"));
p.getChildren().add(title);
p.setLimit("1,1");
return p;
}
```

Nella parte finale del metodo viene recuperato l'oggetto vm istanza di ViewManager, questo oggetto viene subito utilizzato per la memorizzazione dell'oggetto comp istanza di GridBox. Successivamente viene creato un oggetto di tipo ShowInMessageController il quale fungerà da listener degli eventi GridBox.
ACTION_DOUBLE_CLICK. Le ultime righe di codice si occupano di posizionare il GridBox e la Label all'interno del Panel.

BUSINESS LAYER

Finora abbiamo parlato degli aspetti inerenti l'utilizzo delle API messe a disposizione da ThinWire, però non bisogna dimenticare che la nostra applicazione prevede comunque un dominio applicativo ed uno strato di persistenza e di gestione dei dati. Gli utenti dovranno loggarsi quindi dovranno essere forniti almeno di una *username* e di una *password*. Inoltre sarà vantaggioso definire un *id* che identifichi in modo univoco le istanze di tale classe. Riassumendo il tutto la classe UserDTO avrà la seguente struttura

```
public class UserDTO {
  private String id=null;
  private String username=null;
  private String password=null;
  // setters e getters
  . . .
}
```

L'attore principale della nostra applicazione è però rappresentato dalla classe MessageDTO.

```
public class MessageDTO {
  private String id=null;
  private String text=null;
  private String subject=null;
  private Date sentTime=null;
  private UserDTO fromUser=null;
  private UserDTO toUser=null;
  // setters e getters
  . . .
}
```

Anche in questo caso è utile definire un *id* per l'identificazione univoca; ogni messaggio conterrà nel campo *text* il testo del messaggio stesso, mentre nel campo *subject* memorizzerrà il soggetto. Ogni messaggio dovrà tenere traccia anche della data in cui è stato inviato nonchè dell'utente che lo ha inviato e di quello a cui è stato destinato. Ovviamente tali utenti saranno rappresentati da istanze della classe *UserDTO*. Possiamo adesso vedere le caratteristiche dei servizi di

business utilizzati nella nostra applicazione, a tal proposito trascureremo volutamente l'implementazione e lo studio del modo in cui si interagisce col DB. Infatti ci soffermeremo solo sull'analisi della semantica delle interfacce. Le interfacce sono due e rappresentano i servizi inerenti messaggi ed utenti.

```
public interface IUerService {
    public UserDTO login(String username, String
                                          passw);
    public List<UserDTO> listALL();
    public List<UserDTO> listQBE(UserDTO user);
}
```

Per la gestione degli utenti saranno sufficenti tre servizi; il metodo login prende in input username e password e restituisce in caso di successo un oggetto istanza di UserDTO, in caso contrario ritorna null. Il secondo metodo *listALL* restituisce l'elenco completo degli utenti dell'applicazione; il terzo metodo invece esegue una ricerca degli utenti effettuando una Query By Example, ovvero utilizza come filtri per la ricerca i campi valorizzati dell'oggetto user istanza di UserDTO. Vista la natura didattica dell'applicazione non è stato definito un metodo per l'inserimento di nuovi utenti; quelli ammessi all'utilizzo dell'applicazione sono definiti dalle seguenti coppie:

USERNAME	PASSWORD
roby	roby
ale	ale
baggio	baggio
pippo	pippo

Per quanto riguarda i metodi di business inerenti la gestione dei messaggi possiamo notare che il primo metodo effettua una ricerca dei messaggi il cui toUser (ovvero l'utente al quale è stato inviato) ha un *id* uguale a quello passato come parametro.

```
public interface IMessageService {
   public List<MessageDTO> listByToId(String toID);
   public List<MessageDTO> listByFromId(String toID);
   public MessageDTO readById(String meID);
   public String add(MessageDTO msg);
}
```

Analoga funzionalità di ricerca viene effettuata dal metodo listByFromId, in questo caso però la ricerca utilizza come filtro l'id del mittente del messaggio. Il Business Layer espone anche un metodo (readById) che effettua la lettura del messaggio in base al suo id, è ovvio che questa lettura restituisce al più un solo oggetto istanza di MessageDTO. Il metodo add invece effettua il salvataggio su DB del MessageDTO passato come parametro al metodo.

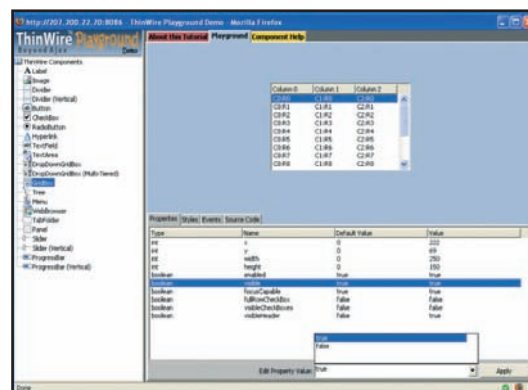



Fig. 3: Playground l'applicazione di auto apprendimento fornita a corredo del framework

I CONTROLLER

Come annunciato in precedenza la nostra applicazione fa un uso intensivo del pattern MVC; quindi non ci resta che capire come funzionano i vari Controller. Vista la natura didattica dell'articolo e lo spazio a disposizione ci limiteremo allo studio di un solo Controller, comunque i principi che regolano il loro funzionamento sono molto simili a quello preso in esame qui di seguito.

```
public class InboxButtonController implements
                                    ActionListener{
   protected ViewManager vm=null;
   public InboxButtonController(ViewManager vm){
     this.vm=vm;
   }
   public void actionPerformed(ActionEvent ev) {
     UserDTO
              me=(UserDTO)WebApp1.session.get().
                                get(Names.USER);
     String myID=me.getId();
     List<MessageDTO>
          list=MessageService.sing().listByToId(myID);
     GridBox gb=(GridBox)this.vm.get(
                       Names.VIEW_GRIDBOX_INBOX);
     Panelp=(Panel)this.vm.get(
                       Names.VIEW_INBOX_PANEL);
     Frame f=(Frame)this.vm.get(
                              Names.VIEW_FRAME);
     Panel p1=(Panel)this.vm.get(
                         Names.VIEW_VISIBLE_PANEL);
     if(p1!=p){
       f.getChildren().remove(p1);
       f.getChildren().add(p);
```

```
            this.vm.put(Names.VIEW_VISIBLE_PANEL,
                                                    p);
        }
        UIFactory.sing().fillGridBox(list, gb, true);
    }
}
```

Il costruttore prende come parametro un ViewManager e lo memorizza per poterlo utilizzare successivamente. Il ruolo principale è però detenuto dal metodo *actionPerformed*, questo metodo viene invocato dalla piattaforma non appena si verifica l'evento per cui il Controller è stato aggiunto come ascoltatore. Nelle prime due righe si occupa di recuperare l'id dell'utente che ha effettuato il login. Con tale stringa (myID) effettua una ricerca utilizzando il *singleton* MessageService; questa ricerca restituisce la lista dei messaggi inviati all'utente. A questo punto carica quattro componenti grafici, precedentemente memorizzati dal ViewManager; nell'ordine carica prima il GridBox da utilizzare per visualizzare l'elenco dei messaggi, successivamente il Panel su cui posizionare il precedente GridBox, e poi il Frame esterno. In ultimo recupera il Panel che attualmente è visualizzato nella zona centrale della pagina web. A questo punto le possibilità sono due: il Panel corrente e quello che contiene i messaggi InBox coincidono oppure sono diversi. Nel primo caso non c'è bisogno di effettuare nessun refresh sul Frame, nel secondo caso invece è necessario rimuovere il Panel corrente, aggiungere quello InBox ed aggiornare il corrente sul ViewManager. Alla fine di queste operazioni è possibile riempire la GridBox gb con la lista dei messaggi recuperati.

CONCLUSIONI

Questo framework AJAX si presenta già pronto per essere utilizzato in applicazioni di medio-piccole dimensioni. Lo abbiamo visto all'opera in un'applicazione relativamente complessa, in cui abbiamo utilizzato alcuni pattern degni di una normale applicazione desktop. La cosa che sicuramente sarà stata notata dal lettore è che il codice utilizzato non presenta nessuna peculiarità tipica delle normali applicazioni web; questo aspetto rende ThinWire un prodotto appetibile ai programmatori Java con trascorsi nello sviluppo di Rich Client Application.

Roberto Sidoti

Roberto Sidoti è Ingegnere Informatico, in passato si è occupato di servizi VoIP; attualmente lavora come Functional Designer per Herzum Software, una multinazionale di consulenza specializzata nello sviluppo di componenti per applicazioni SOA.

CREARE DOCUMENTI WORD SENZA OFFICE

ARRIVA IL NUOVO FORMATO OPENXML. SCOPRIAMO COSA C'È SOTTO E COME POSSIAMO SFRUTTARLO PER INSERIRE NELLE NOSTRE APPLICAZIONI LA POSSIBILITÀ DI SALVARE I DATI IN MODO COMPATIBILE CON OFFICE SENZA DOVER ACQUISTARE L'INTERA SUITE

Microsoft Office 2007 è ormai sul mercato da qualche mese e non è sicuramente passato "inosservato" al vaglio critico della comunità ICT che si è immediatamente divisa tra entusiasti e feroci avversari.
Le novità più evidenti sono senz'altro sul piano dell'interfaccia utente dove il buon vecchio menu è stato sostituito da una discussa interfaccia a schede chiamata Ribbon (cioè *nastro*).

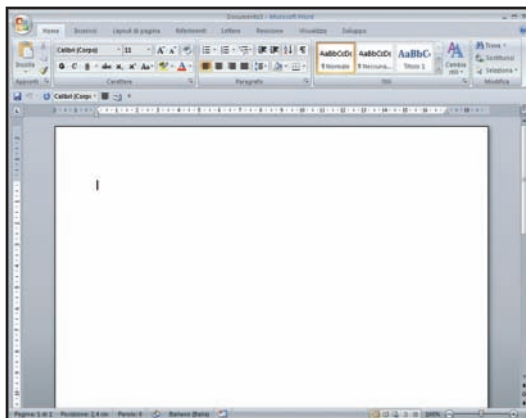



Figura 1: L'interfaccia Ribbon di Office 2007

Inutile cercare il modo di "ripristinare" il menu tipico delle versioni precedenti, Microsoft (con una scelta leggermente "drastica", per usare un eufemismo) ha deciso di mandare definitivamente in pensione la vecchia interfaccia utente a menu!
Se, da una parte, la nuova interfaccia è chiaramente mirata all'utente meno esperto è, dall'altra, decisamente penalizzante per chi è un po' più "navigato".
Comunque la nuova GUI di Office (dettata, secondo me, più da esigenze di marketing che da effettive esigenze) rischia di far passare in secondo piano le vere, importanti novità che sono tutte "sotto il cofano".
Se avete avuto modo di provare Office 2007 vi sarete forse accorti che le estensioni di default dei documenti sono cambiate: non più .doc per Word, ma .docx , non .xls per Excel, ma .xslx e via dicendo.
A prima vista si potrebbe pensare a un cambiamento anch'esso dettato da esigenze di marketing, ma in realtà non è così: è cambiato proprio il formato in cui vengono salvati i documenti!

IL NUOVO FORMATO DI FILE

La cosa più interessante è che adesso i documenti di Office vengono salvati non in formato binario, ma in XML!
Benissimo direte voi ma intanto sarete già andati ad aprire i vari .docx con notepad aspettandovi di vedere i ben noti tag XML!
Calma, la cosa non è così semplice! Descrivere in un file XML un documento con stili, immagini, proprietà ecc... richiede più spazio che farlo in formato binario, c'è da considerare poi che un solo file non basterebbe, ci possono essere le immagini, i file esterni e così via. Per questo in realtà un file di Office 2007 non è altro che un file ZIP rinominato contenente non uno ma diversi file. Ne volete la prova? Facciamo un esperimento.
Creiamo un file di Word salviamolo in una directory con il nome *prova.docx*
Raggiungiamo il file con Esplora Risorse di Windows e rinominiamolo (tasto F2) in *prova.zip*
Se proviamo ad aprire il file ZIP notiamo come all'interno abbiamo una struttura di file e directory di questo tipo:

```
0 _rels
2 .rels
0 docProps
2 app.xml
2 core.xml
0 word
0 _rels
        2 document.xml.rels
```

0 theme
2 theme1.xml
2 document.xml
2 fontTable.xml
2 settings.xml
2 styles.xml
2 webSettings.xml
2 [Content_Types].xml

Come intuiamo dai nomi, ogni file XML descrive una particolare caratteristica del documento: font, stili, impostazioni e contenuto. Già, ma dove il contenuto, cioè il testo che abbiamo editato?
Il contenuto, trattandosi di un documento di word, è in *word/document.xml*, decomprimiamo il file ZIP e apriamo questo file, vedremo un markup di questo tipo:

```
<?xml version="1.0" encoding="UTF-8"
                                    standalone="yes"?>
<w:document
xmlns:w="http://schemas.openxmlformats.org/wordp
                    rocessingml/2006/main">
        <w:body>
                <w:p w:rsidR="002939AA"
                    w:rsidRDefault="00432613">
                        <w:r>
        <w:t>TESTO</w:t>
                        </w:r>
                </w:p>
                <w:sectPr w:rsidR="002939AA">
                        <w:pgSz w:w="11906"
                                    w:h="16838"/>
                        <w:pgMar
    w:top="1417" w:right="1134" w:bottom="1134"
    w:left="1134" w:header="708" w:footer="708"
                                    w:gutter="0"/>
                        <w:cols
                                w:space="708"/>
                        <w:docGrid
                                w:linePitch="360"/>
                </w:sectPr>
        </w:body>
</w:document>
```

Come avrete capito quindi, il contenuto del documento è circondato da un markup di formattazione concettualmente simile all'HTML.

OPENXML

L'insieme di struttura del file ZIP e schema dei file XML è quello che si chiama *Open XML* o, in breve, OOXML (cioè *Office Open XML*).
Open XML, contrariamente agli altri formati utilizzati da Office nelle versioni precedenti, non è un formato proprietario, ma uno standard ECMA registrato a dicembre 2006.
Quali sono le implicazioni di questo nuovo formato, ovvero a cosa ci serve conoscere tutte queste cose?
Pensiamo un attimo ad uno scenario reale : un sito Web propone un servizio in cui l'utente immette alcuni dati e, compilata la form, può scaricare un documento di Word con un contratto precompilato, come lo realizzereste?

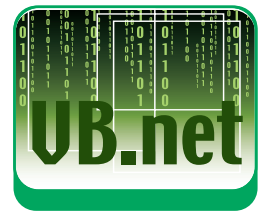
Con le versioni di Office precedenti occorreva:
- installare Office sul server
- usare il modello ad oggetti di Word attraverso la libreria COM per costruire il documento

Questo però comporta notevoli problemi:
- se il sito è in hosting presso un Internet Service Provider ben difficilmente troveremo Office installato
- la libreria COM di Word dà notevoli problemi se usata lato server (tra l'altro, in caso di errore, rimane l'istanza di Word aperta sul server rischiando di esaurire ben presto la RAM), tant'è che la stessa Microsoft ne sconsiglia questo utilizzo

Il nuovo formato invece, non essendo altro che una serie di file XML zippati, consente di comporre dinamicamente il documento (basta scrivere dei file XML appropriati) anche senza usare librerie proprietarie.

MICROSOFT SDK FOR OPENXML FORMATS

Poiché il formato è costituito praticamente solo da XML è possibile anche manipolare e creare documenti di Office anche con i soli strumenti di gestione dell'XML offerti dai vari ambienti di programmazione.
Per semplificare il compito degli sviluppatori, Microsoft ha messo a disposizione un SDK per OOXML (basta cercare su Google "Microsoft SDK for Open XML Formats") che installa nel sistema la libreria che consente di gestire facilmente il formato: *Microsoft.Office.DocumentFormat.OpenXml.dll*.
La libreria (che richiede l'installazione della versione 3 del Framework) fornisce un modello ad oggetti fortemente tipizzato che rappresenta i vari tipi di documento (WordprocessingDocument, SpreadsheetDocument e PresentationDocument).
La libreria non si occupa del contenuto XML del documento, ma di creare e mantenere l'intera struttura di un documento senza dover ri-

correre direttamente a creare i file di default e occuparsi della compressione.

UN PRIMO ESEMPIO

Vediamo di passare subito dalle parole ai fatti. Questa breve classe ci consente di creare un nuovo documento di Word :

```
Imports
 Microsoft.Office.DocumentFormat.OpenXml.Packaging
Imports System.IO
Imports System.Text

Public Class Esempio1

Private Sub CreaNuovoDocumento(ByVal
                                  documentPath As String)
        Dim wordDoc As WordprocessingDocument =
       WordprocessingDocument.Create(documentPath,
                WordprocessingDocumentType.Document)
        Using (wordDoc)
            Dim mainPart As MainDocumentPart =
                         wordDoc.AddMainDocumentPart
            ImpostaContenuto(mainPart)
        End Using
    End Sub

    Private Sub ImpostaContenuto(ByVal part As
                                    MainDocumentPart)
        Const docXml As String = "<?xml
                  version=""1.0"" encoding=""UTF-8""
                         standalone=""yes""?>" & _
        "<w:document
 xmlns:w=""http://schemas.openxmlformats.org/word
                   processingml/2006/main"">" & _
        "<w:body><w:p><w:r><w:t>Hello world!
   </w:t></w:r></w:p></w:body></w:document>"
        Dim stream1 As Stream = part.GetStream
        Dim utf8encoder1 As UTF8Encoding = New
                                       UTF8Encoding()
        Dim buf() As Byte =
                         utf8encoder1.GetBytes(docXml)
        stream1.Write(buf, 0, buf.Length)
    End Sub

End Class
```

La classe non fa altro che creare un nuovo documento di Word 2007 (*WordprocessingDocument*), successivamente va a modificare il contenuto (*MainDocumentPart*) semplicemente andando a scrivere, con il metodo *ImpostaContenuto*, del testo (il classico "Hello world") in formato XML. Andiamo ad eseguire il codice in un progetto Console nel classico modo:

```
Module Module1
    Sub Main()
       Dim obj As New Esempio1
        Dim documentPath As String =
          Path.Combine(Directory.GetCurrentDirectory(),
                                                    ")
        obj.CreaNuovoDocumento(documentPath)
        Console.WriteLine("documento creato in {0}",
                                         documentPath)
    End Sub
End Module
```

Il risultato sarà un documento di Word chiamato "test.docx" creato nella stessa directory del programma. Nell'esempio abbiamo visto però come il contenuto XML del nostro documento abbiamo dovuto scriverlo direttamente nello stream, a cosa serve quindi la libreria dell'SDK di Microsoft?
Beh ... in effetti serve solo a rendere trasparente al programmatore il processo di lettura e scrittura dei dati nel file compresso; la libreria si occuperà di creare la struttura dei file richiesta dallo standard e consentirà di focalizzare il compito nello scrivere i dati del documento.
Pensandoci bene tuttavia il rapporto costi/benefici usando la libreria *OpenXml* non è molto vantaggioso. Questa libreria infatti richiede il .NET Framework 3.0: se sviluppiamo per il Web si trova ancora raramente questa versione sui server dei vari provider, spesso fermi alla 2.0; sviluppando per il desktop invece potrebbe rendersi necessario un aggiornamento dei computer degli utenti.
Tutto questo per un compito, quello svolto dalla libreria, tutto sommato abbastanza semplice: gestire lettura e scrittura in un file compresso. Nulla che non possa essere fatto in altro modo.
Vedremo quindi come creare del codice tutto nostro per leggere e scrivere i file di Office 2007.

GESTIRE DIRETTAMENTE IL FORMATO OPENXML

Senza la libreria dell'SDK di Microsoft dobbiamo in primo luogo occuparci di leggere e scrivere in un file compresso.
Per questo compito possiamo aggiungere ai riferimenti del nostro progetto l'utilissima libreria ICSharpCode.SharpZipLib preferibile, per i nostri scopi, anche alle classi di System.IO.Compression introdotte con il .NET 2.0.
Questa libreria, disponibile anche per la versione 1.1 del Framework, ci consente di leggere e scrivere da e in un file ZIP.
Vediamo, brevemente come facciamo a scrivere dei bytes in un file ZIP:

```
Imports ICSharpCode.SharpZipLib.Zip
```

```
...
    Public Sub AddToZip(ByVal zipFileName As String,
     ByVal entryName As String, ByVal data As Byte())
        Dim out As New
            ZipOutputStream(IO.File.Open(zipFileName,
                          IO.FileMode.OpenOrCreate))
        Dim entry As New ZipEntry(entryName)
        out.PutNextEntry(entry)
        out.Write(data, 0, data.Length)
        out.Finish()
        out.Close()
    End Sub
...
```

e a leggerne il contenuto:

```
    Public Sub ReadZip(ByVal zipFileName As String)
        Dim instream As New
            ZipInputStream(IO.File.Open(zipFileName,
                          IO.FileMode.OpenOrCreate))
        Dim entry As ZipEntry
        While True
            entry = instream.GetNextEntry
            If entry IsNot Nothing Then
                'lettura dell'entry
                Dim size As Integer = 2048
                Dim lb As New List(Of Byte)
                Dim data(size - 1) As Byte
                While True
                    size = instream.Read(data, 0,
                                         data.Length)
                    If size > 0 Then
                        lb.AddRange(data)
                    Else
                        Exit While
                    End If
                End While
                'altre operazioni ...
            Else
                'FINE flusso
                Exit While
            End If
        End While
    End Sub
```

Per i nostri scopi (lettura e scrittura in file Open Xml) predisponiamo quindi due classi di "servizio" *ZipReader* e *ZipWriter*.
In *ZipReader* c'è solo un metodo, *ReadZip*, che ci restituisce il contenuto in forma di *Dictionary*; le chiavi del Dictionary saranno i nomi dei file presenti nel file ZIP, i valori saranno Array di Byte derivanti dalla decompressione del contenuto di questi file. La struttura della classe (non riportiamo qui il codice integrale) sarà quindi :

```
Public Class ZipReader
    Public Function ReadZip(ByVal zipFileName As
                String) As Dictionary(Of String, Byte())
        ...
    End Function
End Class
```

ZipWriter contiene anch'esso un solo metodo, *AddToZip*, ma in tre versioni in modo che sia possibile aggiungere ad un file ZIP più file rappresentati da un Dictionary che ha per chiavi i nomi dei file e per valori l'Array di Byte rappresentanti il contenuto:

```
Public Sub AddToZip(ByVal zipFileName As String,
ByVal dictionaryData As Dictionary(Of String, Byte()))
```

una versione semplificata per aggiungere al file ZIP una sola coppia nome/valore:

```
Public Sub AddToZip(ByVal zipFileName As String,
     ByVal entryName As String, ByVal data() As Byte)
```

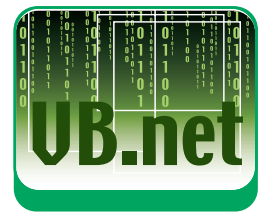
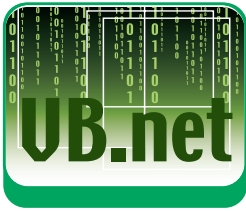
ed infine una versione ancora più semplice della precedente che consente di usare direttamente una stringa al posto della matrice di Byte:

```
Public Sub AddToZip(ByVal zipFileName As String,
     ByVal entryName As String, ByVal data As String)
```

Terminata la creazione del codice di supporto per la gestione del formato ZIP passiamo al nostro generatore di documenti.
La funzione è la stessa di quella che abbiamo vista nel primo esempio : generare un semplice documento di Word con scritta la famosa frase "Hello World". Per essere valido e riconosciuto da Word 2007 il file compresso .docx deve contenere almeno i seguenti file (nelle relative path):

- **_rels/.rels** – in formato XML ed estensione .rels che definisce le relazioni
- **[Content_Types].xml** - che definisce i tipi MIME presenti
- **word/document.xml** – che contiene il testo vero e proprio

Nella classe del nostro esempio impostiamo i contenuti dei tre file come costanti e, nel metodo *Exec*, li aggiungiamo al file *.docx* :

```
Public Class Esempio2
    Const File_Rels As String = "<?xml
  version=""1.0"" encoding=""utf-8""?><Relationships
 xmlns=""http://schemas.openxmlformats.org/packag
                 e/2006/relationships""><Relationship
Type=""http://schemas.openxmlformats.org/officeDoc
            ument/2006/relationships/officeDocument""
                       Target=""/word/document.xml""
     Id=""R90295e44b5f24e8e"" /></Relationships>"
```

RISORSE PER OFFICE XML

Una guida utile a chi vuole addentrarsi nei meandri del dialetto XML che descrive un documento di Office è pubblicata da O'REILLY sotto il titolo di "Office 2003 XML" (applicabile anche a Office 2007). Un capitolo di saggio sul vocabolario di WordProcessingML è disponibile gratuitamente all'indirizzo http://www.oreilly.com/catalog/officexml/chapter/ch02.pdf

```
    Const File_ContentTypes As String = "<?xml version=""1.0"" encoding=""utf-8""?><Types xmlns=""http://schemas.openxmlformats.org/package/2006/content-types""><Default Extension=""xml"" ContentType=""application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml"" /><Default Extension=""rels"" ContentType=""application/vnd.openxmlformats-package.relationships+xml"" /></Types>"
    Const File_Document As String = "<?xml version=""1.0"" encoding=""UTF-8"" standalone=""yes""?><w:document xmlns:w=""http://schemas.openxmlformats.org/wordprocessingml/2006/main""><w:body><w:p><w:r><w:t>Hello world!</w:t></w:r></w:p></w:body></w:document>"
    Public Sub Exec(ByVal filename As String)
        Dim Rels() As Byte = UTF8Encoding.UTF8.GetBytes(File_Rels)
        Dim ContentTypes() As Byte = UTF8Encoding.UTF8.GetBytes(File_ContentTypes)
        Dim Document() As Byte = UTF8Encoding.UTF8.GetBytes(File_Document)
        Dim dictionaryData As New Dictionary(Of String, Byte())
        dictionaryData.Add("_rels/.rels", Rels)
        dictionaryData.Add("[Content_Types].xml", ContentTypes)
        dictionaryData.Add("word/document.xml", Document)
        Dim zw As New ZipWriter
        zw.AddToZip(filename, dictionaryData, ZipWriter.ZipCompression.maximum)
    End Sub
End Class
```

Eseguendo il codice che abbiamo visto otteniamo esattamente lo stesso risultato che abbiamo visto nel primo esempio, ma senza usare le librerie dell'SDK della Microsoft (e quindi eliminando la necessità di disporre di .NET Framework 3).

I DIALETTI XML DI OFFICE

Da quanto abbiamo visto avrete notato che il problema più grosso è semmai districarsi tra il markup XML specifico per Word, Excel ecc... D'altra parte dovete pensare che questo codice deve descrivere il documento di Office in ogni suo minimo particolare : formattazione, colori, stili, temi, posizione e così via. Impararsi WordProcessingML e simili è impresa ardua, non inutile, ma il più delle volte sproporzionata all'obbiettivo: quando mai capiterà di dover comporre ad esempio un intero documento di Word da zero? Molto più probabilmente avremo un template scritto con Word al quale cambiare semplicemente alcune parole segnaposto, in casi simili il nostro programma dovrà fare semplicemente un trova e sostituisci

USO DI UN TEMPLATE

Per vedere subito un esempio pratico di *template* applicato a OpenXml prendiamo in esame un caso abbastanza semplice: abbiamo una tabella di Access (utilizzeremo il nuovo formato di Access 2007) contenente i dati dei clienti, vogliamo comporre una lettera, dato un *template*, prendendo i dati del cliente da una riga della tabella e sostituendoli con i relativi segnaposto del *template*. Un po' come la "stampa unione", per intenderci, solo fatta al di fuori di Word.
Per prima cosa andremo a comporre il *template* creando un normalissimo documento di Word, lì dove vogliamo che vengano inseriti i valori provenienti dalla tabella del database inseriremo dei segnaposto formattati in questo modo:

```
[:nomeCampo]
```

dove naturalmente al posto di "nomeCampo" inseriremo il nome del campo della tabella.
In **figura 2** possiamo vedere il template con i relativi segnaposti.

Ditta Rossi Spa

Spett. le
[:Company]
[:Address]
[:PostalCode][:City]
[:Country]

Oggetto: vendita promozionale prodotto ABC

Spettabile [:ContactTitle][:ContactName],
La nostra azienda è lieta di annunciarle che dal mese di settembre l'articolo ABC sarà messo in vendita con lo sconto del 50%.

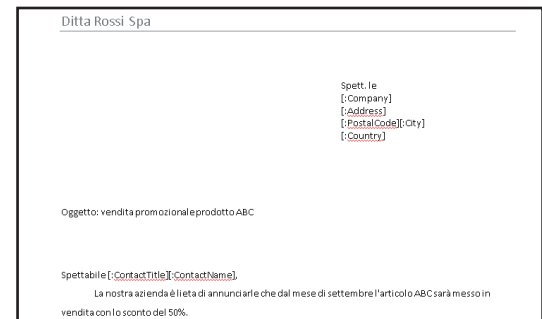



Figura 2: Documento template

Adesso però bisogna effettuare un'altra operazione, modificare manualmente il codice XML del documento:

- Chiudiamo il documento di *Word*
- Rinominiamo il file da *.docx* a *.zip* (ad esempio da *template.docx* a *template.zip*)
- Estraiamo dal file ZIP il documento XML *word/document.xml* aprendolo con un editor di testo
- A questo punto occorre fare attenzione perché probabilmente Word avrà scritto i nostri segnaposti in questo modo:

```
<w:r w:rsidR="00D10817">
<w:t>[:</w:t>
</w:r>
<w:r>
<w:t>Company</w:t>
</w:r>
<w:r w:rsidR="00D10817">
<w:t>]</w:t>
</w:r>
```

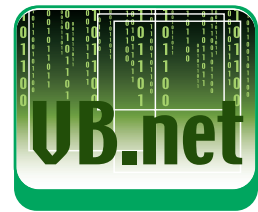
separando cioè i simboli "[:" e "]" dal testo con dei markup aggiuntivi.
Occorre invece semplificare il codice riunendo simboli e testo:

```
<w:r>
<w:t>[:Company]</w:t>
</w:r>
```

- Salviamo il documento XML e reinseriamolo nel file ZIP sostituendolo a quello vecchio, rinominando il file .zip nuovamente in *.docx*.

Questa operazione è necessaria perché effettueremo un "trova e sostituisci" sul codice XML del template e quindi il pattern deve essere contiguo. La nostra funzione di trasformazione aprirà la tabella nel database (cogliamo anche l'occasione per usare la nuova stringa di connessione per i database Access 2007), leggerà la riga del cliente identificato da ID e sostituirà i valori nei rispettivi segnaposti del *template* salvando poi il documento risultante:

```
Imports System.Data.OleDb
Imports System.Text
Public Class Template
    Sub CreaNuovo(ByVal templateFilename As String,
    ByVal IDCliente As String, ByVal docPath As String)
        Dim connString As String =
String.Format("Provider=Microsoft.ACE.OLEDB.12.0;D
            ata Source={0}", "..\..\Northwind.accdb")
        Dim conn As New OleDbConnection(connString)
        Dim cmdString = String.Format("SELECT *
          FROM Customers WHERE CustomerID='{0}'",
                                        IDCliente)
        Dim cmd As New OleDbCommand
                                  (cmdString, conn)
        Dim adp As New OleDbDataAdapter(cmd)
        'acquisiamo dataTable con i dati
        Dim resultTable As New DataTable
        adp.Fill(resultTable)
        If resultTable.Rows.Count = 0 Then
            'la query non ha prodotto risultati
            Throw New Exception
                  ("la query non ha prodotto risultati")
        End If
        Dim zReader As New ZipReader
```

```
        Dim dict As Dictionary(Of String, Byte()) =
                        zReader.ReadZip(templateFilename)
        'acquisisce il contenuto XML
        Dim content As String =
                            UTF8Encoding.UTF8.GetString
                             (dict("word/document.xml"))
        Dim firstRow As DataRow =
                                    resultTable.Rows(0)
        'sostituisce i segnaposti con i valori reali
        For Each col As DataColumn In
                                   resultTable.Columns
            Dim value As String = ""
            If Not firstRow.IsNull(col.ColumnName)
                                                  Then
                value =
                 firstRow.Item(col.ColumnName).ToString
            End If
            Dim bkMark As String = "[:" &
                                  col.ColumnName & "]"
            content = content.Replace(bkMark, value)
        Next
        dict("word/document.xml") =
                   UTF8Encoding.UTF8.GetBytes(content)
        Dim zWriter As New ZipWriter
        zWriter.AddToZip(docPath, dict)
    End Sub
End Class
```

In figura 3 possiamo vedere il risultato della trasformazione.

Ditta Rossi Spa

Spett.le
Alfreds Futterkiste
Obere Str. 57
12209Berlin
Germany

Oggetto: vendita promozionale prodotto ABC

Spettabile Sales Representative Maria Anders,
La nostra azienda è lieta di annunciarle che dal mese di settembre l'articolo ABC sarà messo in vendita con lo sconto del 50%.

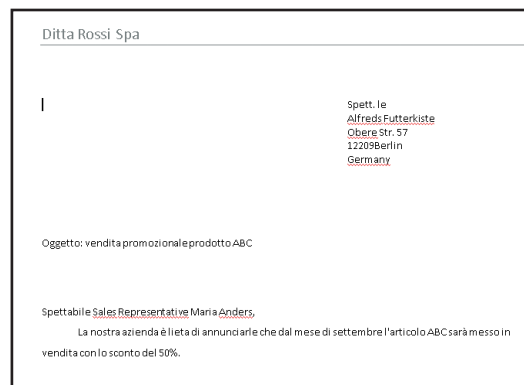



***Figura 3:** il risultato della trasformazione*

CONCLUSIONI

Il formato OpenXML rappresenta uno svolta importante compiuta da MS verso la trasparenza. La sua adozione come standard aperto lo rende particolarmente appetibile anche per la pubblica amministrazione. Comprenderne le basi rappresenta per noi programmatori una straordinaria opportunità.

Francesco Smelzo

WCF E IL MONDO DEI DEVICE CONNESSI

IN UN MONDO DOVE LA DIFFUSIONE DI PERIFERICHE CHE ADOTTANO STANDARD DI COMUNICAZIONE DIVERSI STA FACENDOSI SEMPRE PIÙ CAPILLARE, È IMPORTANTE ADOTTARE SISTEMI CHE GARANTISCONO L'INTEROPERABILITÀ. WCF È UNO DI QUESTI...

L'evoluzione degli attuali strumenti informatici sta seguendo di pari passo le crescenti esigenze della vita di tutti i giorni, anche provocandole in qualche modo. Assistiamo oramai all'uscita quasi quotidiana di nuovi dispositivi che promettono le più mirabolanti tecnologie software spesso a supporto di elementi ludici, quali riproduzioni video o musicali. Personalmente se devo acquistare un dispositivo di questo genere penso al supporto e alla connettività che mi viene offerta. È importante che il mio dispositivo sia in grado di connettersi, ad esempio, ad Internet con le più comuni tecnologie, penso al Wi-Fi. Questo perché potrei avere l'esigenza di farci girare sopra un software capace di trasmettere qualsiasi dato verso, ad esempio, un mio server aziendale. Potrei un giorno avere la necessità di raccogliere i miei appuntamenti e di inviarli contestualmente alla mia azienda dove, in tempo reale, il mio tempo viene impostato come *Occupato* oppure dove, sempre in tempo reale, mi viene segnalato già un altro appuntamento nello stesso orario. Con l'avvento del nuovo .NET Compact Framework 3.5 siamo ora in grado di sviluppare applicazioni SOA con l'utilizzo di una versione ridotta di Windows Communication Foundation (WCF) e che rispecchiano, come vedremo in questo articolo, le caratteristiche di una applicazione come quella descritta.

PROGETTO (DIAGRAMMI UML)

Il progetto che realizzeremo con questo articolo prevede proprio la possibilità di centralizzare tutto l'archivio degli appuntamenti e renderlo facilmente fruibile da qualsiasi applicazione o dispositivo che voglia interagire con esso. Analizziamo l'architettura della nostra applicazione per indagare sulla distribuzione delle diverse componenti. Questo passaggio è fondamentale se vogliamo realizzare una robusta e funzionale applicazione Service-Oriented. Lato server avremo, perciò, le componenti che forniranno i servizi di registrazione e di interrogazione degli appuntamenti. Individuiamo pertanto una componente *AppuntamentoService* che espone due operazioni: RegistraAppuntamento e RecuperaAppuntamenti.

Semplificando di molto le caratteristiche di una applicazione di questa portata, possiamo riassumere le nostre esigenze nella necessità di registrare un appuntamento da un client ed eventualmente interrogare una lista di appuntamenti filtrata per utente e/o per data per data.

Utilizzando un diagramma UML disegniamo il nostro componente figura1:

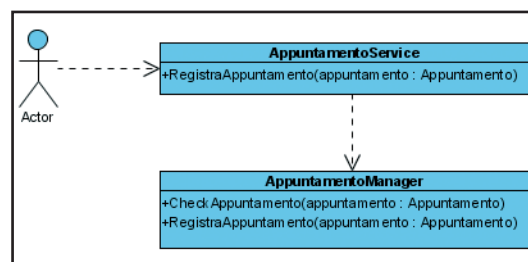



Figura 1: Il Communication Diagram del nostro componente

SERVIZIO WCF

Il primo passo è quello di creare il progetto lato server che si occupa di fornire il set di servizi consumati dai differenti client.

La fruibilità del servizio è una caratteristica fondamentale della nostra architettura e la sua interoperabilità ne è la chiave. Per ottenere questo risultato il *basicHttpBinding* è la tipologia di binding che in WCF offre il più elevato livello di interoperabilità verso qualsiasi client. Inoltre, allo stato attuale dello sviluppo delle librerie WCF per .NET Compact Framework 3.5, questo è il binding più facilmente utilizzabile per quello che andremo a vedere più avanti.

Dopo aver avviato la virtual machine e Visual Studio preinstallato, creiamo una nuova solution AgendaSolution. Ora aggiungiamo un nuovo progetto ioProgrammo.Agenda.Services che sarà il contenitore del codice dei nostri servizi. Il nuovo progetto utilizzerà il relativo template WCF Service Library, come riportato in figura 2.

Conoscenze richieste
Conoscenze base di programmazione in C#

Software
Visual Studio Orcas June 2007 CTP, Virtual PC

Impegno

Tempo di realizzazione

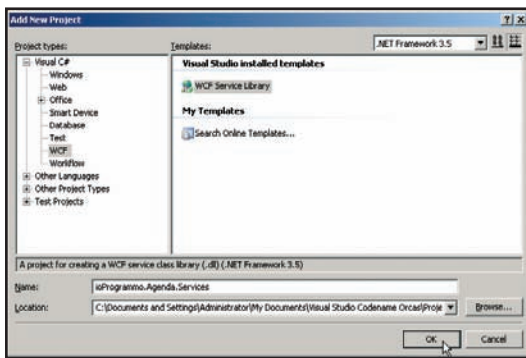



Figura 2: Creiamo il progetto basato sul template WCF Service Library

La forma più corretta e più generalmente accettata in WCF per esprimere un contratto è l'utilizzo, nel codice, delle interfacce. Questo perché il concetto di interfaccia in sé esprime proprio un contratto. Perciò anche noi nel nostro progetto creiamo l'interfaccia per dichiarare il contratto che i nostri servizi dovranno rispettare. Aggiungiamo, quindi, al progetto la seguente interfaccia:

```
[ServiceContract]
[XmlSerializerFormat()]
public interface IAppuntamentoService
{
    [OperationContract]
    string RegistraAppuntamento(Appuntamento
                                        intParam);
    [OperationContract]
    Appuntamento[] GetAppuntamenti();
}
```

L'interfaccia viene decorata con l'attributo *ServiceContractAttribute* che serve per dichiarare che quella specifica interfaccia definisce un contratto per un servizio WCF. Ogni metodo viene invece decorato con un attributo *OperationContractAttribute* che dichiara le varie operation esposte dal nostro servizio. Inoltre utilizzando l'attributo *XmlSerializerFormatAttribute* dichiariamo esplicitamente di voler utilizzare l'*XmlSerializer* per la serializzazione e la deserializzazione delle classi scambiate dal nostro servizio. Questo perchè, allo stato attuale, non è ancora disponibile il *DataContractSerializer* nella versione di WCF per Compact Framework. Dopo aver dichiarato l'interfaccia del servizio, passiamo ora alla sua implementazione. Il codice è molto semplice e possiamo realizzare il nostro servizio in questo modo:

```
public class AppuntamentoService :
                            IAppuntamentoService
{
    public string
        RegistraAppuntamento(Appuntamento intParam)
    {
        AppuntamentiManager.Save(intParam);
        return "registrato";
    }
    public Appuntamento[] GetAppuntamenti()
    {
        return AppuntamentiManager.Read();
    }
}
```

Omettiamo, per motivi di spazio, il codice che compone la classe *AppuntamentiManager*, reperibile sul cd allegato alla rivista, e vediamo come possiamo esporre il nostro servizio. I requisiti della nostra applicazione richiedono che il servizio debba essere raggiungibile da qualsiasi tipologia di client. Per questo scopo abbiamo scelto l'uso del protocollo *http* e, come già specificato, il relativo binding *basicHttpBinding*. Scegliamo quindi di utilizzare come host del nostro servizio un server HTTP, come quello integrato in Visual Studio. Aggiungiamo alla solution un nuovo progetto Web, selezioniamo il template WCF Service, indichiamo File System come location ed impostiamo *AgendaServices* come nome del progetto. Le impostazioni sono riportate in figura3.

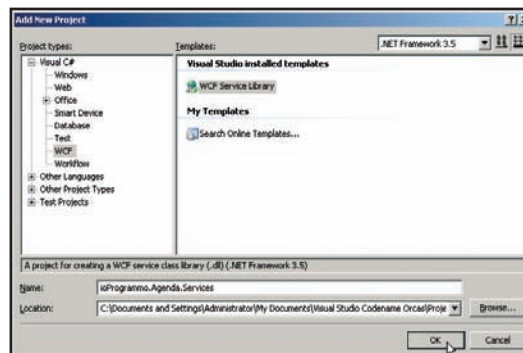



Figura 3: Creiamo il progetto web che ospiterà il nostro servizio

Contestualmente al progetto vengono create le pagine *Service.svc* e *Service.cs*. Rimuoviamo dal progetto la seconda, che si trova nella cartella *App_Code*, e rinominiamo la prima in *AppuntamentoService.svc*. Successivamente apriamo in modifica il file e modifichiamo l'intestazione con la seguente:

```
<%@ServiceHost Language="C#" Debug="true"
        Service="ioProgrammo.Agenda.Services.
                AppuntamentoService" %>
```

Questo ci consente di impostare come implementazione del servizio la classe *AppuntamentoService*. Ora aggiungiamo un riferimento al progetto *ioProgrammo.Agenda.Services* precedentemente creato. Infine eseguiamo l'ultimo passaggio che consiste nella definizione della configurazione del nostro servizio. Apriamo il file web.config e sostituiamo la sezione

<system.serviceModel> con questa:

```
<system.serviceModel>
  <services>
    <servicename="ioProgrammo.Agenda.Services.
                                AppuntamentoService"
behaviorConfiguration="AppuntamentoService
                                        Behavior">
      <endpoint
contract="ioProgrammo.Agenda.Services.
                                IAppuntamentoService"
binding="basicHttpBinding"/>
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior
              name="AppuntamentoServiceBehavior">
        <serviceMetadata httpGetEnabled="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

in questo modo dichiariamo che il servizio *AppuntamentoService* utilizza il binding *basicHttpBinding*. Non abbiamo la necessità dei definire nessun indirizzo poichè, nel caso di servizio esposto su un web server esterno, esso ci viene fornito dal nostro host. Con questo passaggio abbiamo terminato la configurazione del servizio. Resta da definire la classe *Appuntamento*, contenitore dei dati e che può essere rappresentata come di seguito:

```
[SerializableAttribute()]
[XmlTypeAttribute(Namespace =
            "http://Microsoft.ServiceModel.Samples")]
public class Appuntamento
{
    [XmlElementAttribute(Order = 0)]
    public string idAppuntamento;
    [XmlElementAttribute(Order = 1)]
    public string descrizione;
    [XmlElementAttribute(Order = 2)]
    public DateTime orario;
    [XmlElementAttribute(Order = 3)]
    public TimeSpan durata;
}
```

VISUAL STUDIO ORCAS JUNE 2007 CTP

È possibile scaricare la Virtual Machine utilizzata in questo articolo direttamente all'indirizzo: http://download.microsoft.com/download/f/2/a/f2ac411f-acf9-42a7-a84f-3efc409bcd6b/VSTS_VPCJuneCTP.mht. **È anche possibile trovare le novità di Visual Studio al seguente indirizzo:** http://msdn2.microsoft.com/en-us/vstudio/aa700830.aspx

ora il nostro servizio è pronto per essere utilizzato dai client che vorranno connettersi.

CLIENT MOBILE

Preparata l'applicazione lato server, ora ci occupiamo di esplorare e verificare l'effettiva interoperabilità con un client sviluppato su piattaforma .NET Compact Framework 3.5 (CF). Cerchiamo di capire come realizzare il nostro client sfruttando le poche risorse che il framework attuale ci mette a disposizione. Aggiungiamo alla soluzione un nuovo progetto di tipo Mobile chiamandolo ioProgrammo.Agenda.Mobile.
Ora modifichiamo il form di partenza introducendo i vari controlli fino ad ottenere un'interfaccia simile a quella riportata in figura 4.
Per poter interagire con il servizio remoto dobbiamo ora creare il proxy client il cui compito sarà proprio

Figura 4: Il form del nostro client mobile.

quello di smistare le chiamate verso il servizio e di mettersi in ascolto per ricevere le risposte. Allo stato attuale dello sviluppo, utilizzando la June 2007 CTP di Visual Studio, è possibile generare un proxy client automaticamente, è decisamente inutilizzabile a causa della mancanza del DataContractSerializer per il Compact Framework.
Creiamo, quindi, la classe *Appuntamento* così come definita lato server nell'ultima parte del precedente paragrafo ed infine creiamo una nuova classe *Ap-*

puntamentoServiceProxy nella quale definiamo il metodo *RegistraAppuntamento*. Come unico parametro del metodo abbiamo, appunto, un oggetto di tipo *Appuntamento*. Il codice implementato in questo metodo è un pò più complesso ed andremo ad analizzarlo passo per passo.
La mancanza del *DataContractSerializer* ci costringe a sviluppare un serializer ad-hoc per la risoluzione delle problematiche di serializzazione e deserializzazione dei messaggi. Vedremo perciò come implementare manualmente queste componenti che sono fondamentali per il funzionamento del nostro client. Il primo passo da fare è quello di creare una classe lato client il cui compito è semplicemente la replica del messaggio da inviare al server. Questo significa che i tipi scambiati devono rispecchiare quanto stabilito nel WSDL. Lo schema dei parametri del WSDL indica così l'elemento *RegistraAppuntamento*:

```
<xs:element name="RegistraAppuntamento">
    <xs:complexType>
        <xs:sequence>
            <xs:element
                minOccurs="0" maxOccurs="1" name="intParam"
                type="q1:Appuntamento"
                xmlns:q1="http://Microsoft.ServiceModel.Samples"
            />
        </xs:sequence>
    </xs:complexType>
</xs:element>
L'equivalente in codice è il seguente:
[SerializableAttribute()]
[XmlTypeAttribute(Namespace ="http://tempuri.org/")]
public class RegistraAppuntamento
{
    [XmlElementAttribute()]
    public Appuntamento intParam;
}
```

nel codice del nostro metodo *RegistraAppuntamento* creiamo un'istanza della richiesta, valorizziamo il parametro di input e creiamo il messaggio utilizzando il nostro serializer personalizzato:

```
RegistraAppuntamento registraAppuntamento =
        new RegistraAppuntamento();
registraAppuntamento.intParam = app;
XmlSerializerWrapper wrapper =
    new
    XmlSerializerWrapper(typeof(RegistraAppuntamento),
        "RegistraAppuntamento", "http://tempuri.org/");
Message m =
    Message.CreateMessage(MessageVersion.Soap11,
"http://tempuri.org/IAppuntamentoService/Registra
        Appuntamento",
    registraAppuntamento, wrapper);
```

a questo punto istanziamo il binding da utilizzare, il *basicHttpBinding*, e da questo creiamo il nostro *ChannelFactory*:

```
BasicHttpBinding binding = newBasicHttpBinding();
BindingParameterCollection parameters = new
        BindingParameterCollection();
IChannelFactory<IRequestChannel>channelFactory =
    binding.BuildChannelFactory<IRequestChannel>(para
        meters);
channelFactory.Open();
```

Ora istanziamo il nostro channel definendo l'address da utilizzare:

```
EndpointAddress address =
    new EndpointAddress(
        new
        Uri("http://OrcasBeta1VSTS/AgendaServices/
            AppuntamentoService.svc"));
IRequestChannel outChannel =
        channelFactory.CreateChannel(address);
outChannel.Open();
```

infine, inviamo la richiesta:

```
Message reply = outChannel.Request(m);
```

catturiamo la risposta deserializzandola semplicemente utilizzando un XmlDictionaryReader:

```
XmlDictionaryReader bodyReader =
        reply.GetReaderAtBodyContents();
string result =
    Convert.ToBoolean(bodyReader.ReadString());
bodyReader.Close();
```

ed il gioco è fatto. Nel codice illustrato, infatti, la responsabilità della serializzazione della richiesta e della deserializzazione della risposta è completamente a carico nostro. Il framework ci mette a disposizione esclusivamente le classi per l'accesso al messaggio. Abbiamo volutamente escluso dall'articolo l'implementazione della classe *XmlSerializerWrapper* che è comunque presente nel progetto sul CD allegato alla rivista.
L'implementazione del servizio di recupero della lista di appuntamenti è molto simile e potete trovare il codice completo sul CD allegato alla rivista.

CONCLUSIONI

Il nuovo supporto per il .NET Compact Framework, colma una vistosa lacuna. Di sicuro la strada intrapresa è buona ma soffre, come abbiamo visto, ancora di molte mancanze. Gli elementi di studio, però, sono già a nostra disposizione e sicuramente n on mancano i metodi per aggirare i problemi illustrati

Fabio Cozzolino

L'AUTORE

Fabio Cozzolino lavora come analista sviluppatore nello sviluppo di progetti web ed applicazioni distribuite presso la Fimesan Srl di Molfetta (BA). È cofondatore e presidente di DotNetSide, lo user group del sud italia il cui intento è quello di organizzare eventi di maggior spessore tecnico legati allo sviluppo con il .NET Framework. Possiede la certificazione MCAD.NET. Il suo blog è raggiungibile all'indirizzo http://www.dotnetside.org/blogs/fabio

GOOGLE DESKTOP FATTO DA TE

COSTRUIAMO UN MOTORE DI INDICIZZAZIONE DI FILE UTILIZZANDO LA REFLECTION PER INVOCARE DINAMICAMENTE I METODI DI PARSING SULLA BASE DELL'ESTENSIONE DEL FILE DA PROCESSARE. IN QUESTO MODO RENDEREMO IL NOSTRO SOFTWARE MODULARE

Dobbiamo realizzare un software che ci permetta di indicizzare il contenuto dei file presenti sul nostro hard disk. Per farlo, abbiamo la necessità di includere nel programma dei moduli capaci di processare i file e leggerne il contenuto.
Una volta disegnata la logica applicativa, dobbiamo scrivere i moduli per il parsing dei file. Questa è sicuramente l'operazione più onerosa dal punto di vista del tempo poiché i tipi di file che occorre processare sono moltissimi.
Abbiamo però un problema, anche se per la prima release non occorre che il programma sia in grado di processare tutti i tipi di file ma solo tre formati (txt, doc, rtf) e dobbiamo consegnare il lavoro in due giorni. Altri parser verranno richiesti in seguito, magari uno o due al giorno.
Quello che ci preme è realizzare un software modulare e "pluggabile" che ci consenta di aggiungere nuovi moduli per il parsing di tipo di documento che non avevamo previsto quando abbiamo rilasciato il software. Poiché gli aggiornamenti saranno frequenti, vorremmo scrivere il programma in modo che non occorra effettuare un rilascio di tutto il software ad ogni aggiunta di parser, ma ci basti ad esempio effettuare il deploy di un file in una cartella specifica.

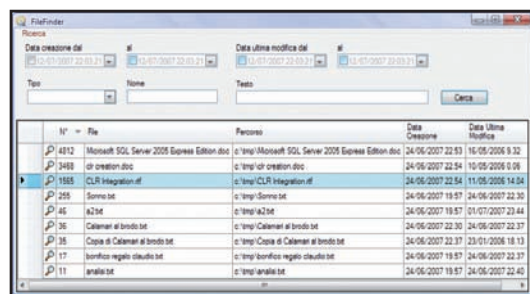



Figura 1: *Il Finder in azione*

L'esecuzione dinamica del codice è uno strumento molto potente da utilizzare in questi e molti altri casi. Il *Framework* .Net fornisce degli strumenti che ci consentono di creare soluzioni eleganti e flessibili basate sull'esecuzione dinamica del codice.

LA REFLECTION

Per *Reflection* si intende la possibilità di recuperare informazioni sugli oggetti a *runtime*. Possiamo sfruttare le informazioni recuperate in vari modi: eseguendo metodi, leggendone le proprietà ed i valori, solo per citare alcuni esempi.
Alla base di tutto sta il fatto che il codice scritto per il *Framework* .*Net* "riflette" ovvero descrive se stesso tramite i metadati generati dal compilatore.
Utilizzando la *Reflection* possiamo reperire informazioni sui tipi contenuti in un *Assembly*, istanziare dinamicamente delle classi o eseguire dei metodi. Possiamo altresì utilizzare la *Reflection* per generare del codice a *runtime*, compilarlo ed eseguirlo.

CARICHIAMO UN ASSEMBLY

Per un primo esempio sulla *Reflection*, creiamo una semplice classe *Person* con un costruttore e due metodi.

```
Public Class Person

    Private _Name As String
    Private _Surname As String
    Private _Age As Integer

    Public Sub New(ByVal Name As String, ByVal
            Surname As String, ByVal _age As Integer)
        _Name = Name
        _Surname = Surname
        _age = Age
    End Sub

    Public Function Print() As String
        Return "Nome: " & _Name & "; Cogome: " &
                        _Surname & "; Età: " & _Age
    End Function

    Public Sub UpdateAge(ByVal NewAge As Integer)
        _Age = NewAge
    End Sub
```

```
End Class
```

Inseriamo la classe nel progetto "*PersonsDemo*" che referenzieremo nel progetto principale "*Reflection-Samples*".
La prima cosa da fare quando si vuole utilizzare la *Reflection* è caricare in memoria l'*Assembly* che si vuole utilizzare. Possiamo farlo con metodi diversi. Nel nostro esempio - di cui trovate il codice completo nella cartella *Samples* del CD allegato – avendo referenziato l'Assembly nel progetto principale, possiamo caricare quest'ultimo in modo semplice, chiamandolo per nome.

```
Dim asm As Assembly
asm = Assembly.Load("PersonsDemo")
```

È anche possibile caricare un *Assembly* indicando il percorso fisico su cui esso si trova.

INTERROGHIAMO L'ASSEMBLY

Una volta caricato in memoria l'*Assembly*, possiamo accedere ai tipi che ne fanno parte:

```
Dim tps() As Type = asm.GetTypes()

For Each t1 As Type In tps
        Console.WriteLine("Tipo trovato: " &
                                              t1.FullName)
Next
```

Tramite il metodo ***GetTypes*** otteniamo un *Array* di *Type* che possiamo scorrere tramite un ciclo *For Each*.
Se si conosce il nome del tipo da caricare, è possibile richiamarlo direttamente tramite il metodo ***GetType***:

```
Dim t2 As Type =
                    asm.GetType("PersonsDemo.Person")
Console.WriteLine("Tipo trovato: " & t2.FullName)
```

L'ESECUZIONE DINAMICA

Una volta che abbiamo recuperato le informazioni sui tipi di un *Assembly*, possiamo operare su di essi. È possibile ad esempio richiamare il costruttore di una classe oppure invocare un metodo specifico. Vediamo come:
In sequenza: istanzieremo un oggetto di tipo Person, ne stamperemo a video i contenuti, modificheremo una proprietà e ristamperemo il contenuto modificato.

```
'Leggo il tipo
Dim PersonType As Type =
                    asm.GetType("PersonsDemo.Person")

'Definisco e valorizzo i parametri dei costruttore
Dim ConstructorTypes() As Type = { _
        GetType(System.String), _
        GetType(System.String), _
        GetType(System.Int32)}
Dim ConstructorParams() As Object = { _
        "Carmelo", _
        "Scuderi", _
        34}

'Leggo il costruttore
Dim Constructor As ConstructorInfo
Constructor =
        PersonType.GetConstructor(ConstructorTypes)

'Invoco il costruttore
Dim Pers As Object =
              Constructor.Invoke(ConstructorParams)
```

Dopo aver letto il tipo, dichiariamo i parametri del costruttore e li valorizziamo tramite i due *Array ConstructorTypes* e *ConstructorParams*.
Otteniamo le informazioni del costruttore tramite il metodo ***GetConstructor*** a cui passiamo i tipi di parametro che ci aspettiamo di trovare ed infine lo invochiamo tramite il metodo ***Invoke*** a cui passiamo l'*Array* contenente i valori dei parametri.
Ottenuto l'oggetto, possiamo eseguirne i metodi. Come prima cosa, proviamo a lanciare il metodo Print che restituisce il contenuto dell'oggetto in formato stringa:

```
'Istanzio ed invoco il metodo Print
Dim Print As MethodInfo =
                    PersonType.GetMethod("Print")
Dim PersInfo As String = Print.Invoke(Pers, Nothing)
```

Abbiamo utilizzato il metodo ***GetMethod*** su *PersonType* per ottenere un oggetto di tipo ***MethodInfo***; su esso eseguiamo il metodo ***Invoke*** utilizzando come parametro l'oggetto *Pers* istanziato dinamicamente.
Possiamo anche eseguire dei metodi con parametro.
Analizziamo le seguenti righe di codice:

```
'Dichiaro e valorizzo i parametri del metodo
Dim MethodTypes() As Type =
                    {GetType(System.Int32)}
Dim MethodParams() As Object = {35}

'Istanzio ed invoco il metodo UpdateAge
Dim Update As MethodInfo =
   PersonType.GetMethod("UpdateAge", MethodTypes)
Update.Invoke(Pers, MethodParams)
```

Per recuperare l'oggetto ***MethodInfo*** utilizziamo un

IL CODICE ALLEGATO

Gli esempi nell'articolo sono in Visual Basic; per chi fosse interessato, nel CD allegato è presente tutto il codice sorgente anche in C#.

Array che definisce la lista dei parametri e, quando lanciamo il metodo ***Invoke***, li valorizziamo tramite l'*Array* "*MethodParams*"

COSTRUIAMO L'INDEXER

Grazie alle nozioni di base sulla Reflection che abbiamo appena esposto possiamo passare alla realizzazione del nostro motore di indicizzazione.

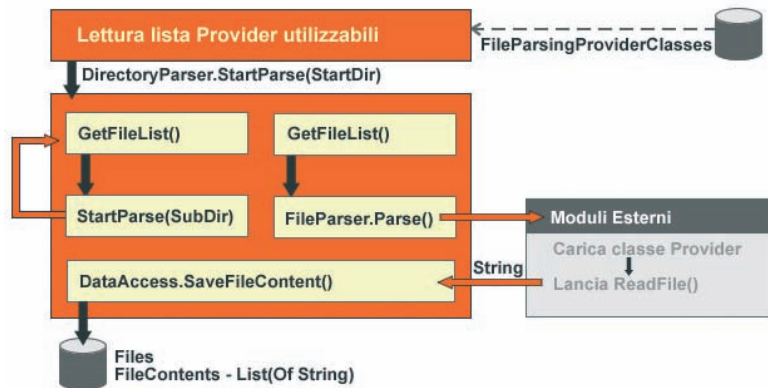



Figura 2: Lo schema del programma

Il programma fornito nel CD come esempio è un eseguibile *Console*; chiaramente è possibile utilizzare il codice fornito per realizzare un servizio Windows in modo da automatizzare le operazioni di indicizzazione.
La prima operazione è la lettura dei *Provider* implementati.
Il programma legge un parametro in ingresso rappresentante una directory ed inizia a processare i file da quest'ultima.
La procedura di lettura file è ricorsiva; per ogni directory processata vengono eseguite due operazioni:

- parsing dei file
- parsing di tutte le sottodirectory.

Il parsing dei file viene fatto utilizzando la Reflection in quanto i moduli che permettono di leggere il contenuto dei file sono esterni al programma.
Per ogni file viene caricata la classe provider ed eseguito il metodo ReadFile() che restituisce una stringa rappresentante il contenuto del file.
Una volta letto il contenuto, i dettagli vengono memorizzati su database.

METADATA

I Metadata sono dei dati che descrivono i tipi .Net; classi e metodi del namespace System.Reflection si basano su di essi per poter recuperare a runtime le informazioni su Assembly, tipi, classi, proprietà e metodi.

IL DATABASE

Analizziamo brevemente la struttura del database che utilizzeremo.
Le tabelle in cui salveremo i file ed il loro contenuto sono rispettivamente Files e FileContents.

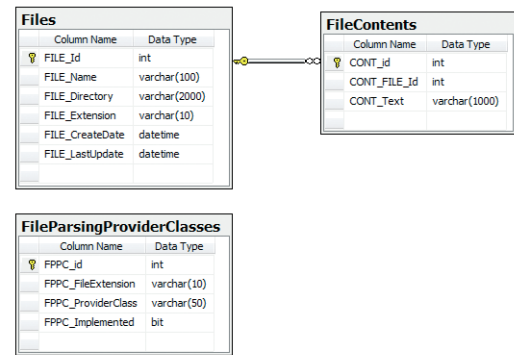



Figura 3: Le tabelle del database

La tabella ***FileParsingProviderClasses*** contiene le informazioni sui provider installati; in particolare, in base all'estensione del file, ci dice qual è l'Assembly da utilizzare e se la classe di parsing è stata implementata.

Table - dbo.FileParsingProviderClasses | Summary

FPPC_id	FPPC_FileExtension	FPPC_ProviderClass	FPPC_Implemented
1	txt	TextFileParser.TextFileParser	True
2	pdf	PdfFileParser.PdfFileParser	False
3	xls	XlsFileParser.XlsFileParser	False
4	doc	DocFileParser.DocFileParser	True
5	rtf	DocFileParser.DocFileParser	True
NULL	*NULL*	*NULL*	*NULL*

Figura 4: Le classi provider

Notiamo che il campo *FPPC_ProviderClass* è formattato come ***NomeNamespace.NomeClasse***.
Passiamo adesso ad analizzare il codice nel dettaglio.
La nostra prima classe, ***ParsedFileInfo***, rappresenta un file del nostro HardDisk. Per brevità, omettiamo le proprietà readonly della classe, comunque presenti nel CD allegato.

```
Public Class ParsedFileInfo

    Private _Directory As String
    Private _Name As String
    Private _Extension As String
    Private _CreateDate As DateTime
    Private _LastUpdate As DateTime
    Private _ParsedContent As List(Of String)

    Public Sub New( _
        ByVal Directory As String, ByVal Name As String, ByVal
                                        Extension As String, _
        ByVal CreateDate As DateTime, ByVal LastUpdate As
                                                  DateTime)

        _Directory = Directory
        _Name = Name
        _Extension = Extension
        _CreateDate = CreateDate
        _LastUpdate = LastUpdate
```

```
    End Sub

End Class
```

Un file è rappresentato da:

- *Directory*: la cartella in cui si trova
- *Name*: il nome
- *Extension*: l'estensione
- *CreateDate*: la data di creazione
- *LastUpdate*: la data di ultima modifica
- *ParsedContent*: il suo contenuto rappresentato da una lista di parole.

La classe ***DataAccess*** si occupa di dialogare con il database. Ne tralasciamo i dettagli per brevità; il codice completo è comunque presente nel CD allegato.
Il cuore dell'applicazione risiede in due classi:

- ***DirectoryParser***: si occupa di leggere i file e scorrere le directory
- ***FileParser***: si occupa di processare il singolo file.

Vediamole una alla volta.
La classe *DirectoryParser* ha due metodi:
Il metodo ***StartParse***

```
[...]

Dim pfiList As List(Of ParsedFileInfo)
pfiList = GetFileList(StartDir)

For Each pfi As ParsedFileInfo In pfiList

    Try

        If
(_FileParserClass.ContainsKey(pfi.Extension.ToLower(
                                                    ))) Then

            Dim fileState As Integer =
                DataAccess.GetFileState(Connection, pfi)

            If (fileState = 0 Or fileState = 2) Then

                Dim fileContent As List(Of String) = _
                    FileParser.Parse(_FileParserClass, pfi)

                If (fileContent.Count > 0) Then

                    pfi.ParsedContent = fileContent

                    If (fileState = 2) Then
                        DataAccess.DeleteFile(Connection,
                                                      pfi)
                    End If
```

```
DataAccess.SaveFileContent(Connection, pfi)
                End If

            End If

            Log.WriteLog("Processato file: " & _
pfi.Name & " (status " & fileState & ")")

        End If

    Catch ex As Exception
        Log.WriteLog("Impossibile processare il file: " &
                                                 pfi.Name)
        Log.WriteLog("Errore: " & ex.Message)
    End Try
Next

For Each subDir As DirectoryInfo In
                                  dir.GetDirectories()
    'Eseguo il parse sulle sottodirectory
    StartParse(subDir.FullName)
Next

[...]
```

Legge i file contenuti nella directory che si sta processando e per ciascuno di essi:

- verifica se è presente nel database
- se non è presente lo salva
- se è presente ma è stato modificato cancella i dati del vecchio e lo salva
- se è presente e non modificato va avanti senza effettuare alcuna operazione.

Il metodo ***GetFileList*** non fa altro che scorrere la directory ed aggiungere i file trovati ad una lista *pfiList* di file da processare.
Possiamo notare che il contenuto del file viene letto tramite il metodo ***Parse*** della classe ***FileParser***.

```
Public Shared Function Parse( _
    ByVal FileParserClass As Dictionary(Of String,
                                          String), _
    ByVal pfi As ParsedFileInfo) As List(Of String)

    Dim processedContent As List(Of String) = New
                                     List(Of String)()
    Dim baseContent() As String

    Dim parserClass As String =
            FileParserClass(pfi.Extension.ToLower())

    Dim Asm As Assembly
    Dim t As Type

    'Carico l'assembly
    Try
```

SYSTEM. REFLECTION

Il Namespace System.Reflection contiene classi ed interface per leggere a runtime le informazioni sugli Assembly .Net L'elenco completo delle classi e molti esempi all'indirizzo: http://msdn2.microsoft.com/it-it/library/system.reflection(VS.80).aspx

```
            Asm = Assembly.LoadFrom( _
                Directory.GetCurrentDirectory() & _
                "\Parsers\" & _
                parserClass.Split(".")(0) & ".dll")

            If (Not Asm Is Nothing) Then
                t = Asm.GetType(parserClass)
            Else
                Throw New Exception("Modulo di parsing
                                                non trovato")
            End If

            If (t Is Nothing) Then
                Throw New Exception("Classe di parsing non
                                                implementata")
            End If

        Catch

            Throw New Exception("Modulo di parsing non
                                                trovato")
        End Try

        'Carico il metodo
        Dim types() As Type = {GetType(String)}
        Dim param() As String = {pfi.Directory + "\" &
                                                pfi.Name}
        Dim m As MethodInfo = t.GetMethod("ReadFile",
                                                types)

        'Eseguo il metodo
        If (Not m Is Nothing) Then
            Try
                Dim content As String =
                    CType(m.Invoke(Nothing, param), String)

                'scarto le parole inferiori a 4 caratteri
                baseContent = content.Split(" ")
                For Each x As String In baseContent
                    If (x.Length > 3) Then
                        processedContent.Add(x)
                    End If
                Next
            Catch
                Throw New Exception("Impossibile
                                processare il file" & pfi.Name)
            End Try
        Else
            Throw New Exception("Metodo Parse non
                                                trovato")
        End If

        Return processedContent

    End Function
```

Dopo aver istanziato una lista di stringhe che dovranno contenere il testo del file processato, leggiamo la classe di parsing relativa al file in questione.
Carichiamo quindi l'*Assembly* relativo all'estensione:

```
Asm = Assembly.LoadFrom( _
            Directory.GetCurrentDirectory() & _
            "\Parsers\" & _
            parserClass.Split(".")(0) & ".dll")
```

Stiamo supponendo che le librerie per il parsing vengano caricate nella cartella ***Parsers*** dell'applicazione. Il secondo passo è leggere la classe che effettua il parsing

```
t = Asm.GetType(parserClass)
```

La stringa *parserClass* è la coppia ***Namespace.Classe*** ottenuta leggendo il campo *FPPC_ProviderClass* della tabella *FileParsingProviderClasses*.
Caricata la classe, non ci resta che recuperare il metodo ed eseguirlo. Cerchiamo un metodo chiamato ***ReadFile*** che accetta come parametro un stringa.

```
Dim types() As Type = {GetType(String)}
Dim m As MethodInfo = t.GetMethod("ReadFile",
                                        types)
```

Infine, valorizziamo i parametri ed eseguiamo il metodo tramite ***Invoke***.

```
Dim param() As String = {pfi.Directory + "\" &
                                        pfi.Name}
Dim content As String = CType(m.Invoke(Nothing,
                                param), String)
```

Abbiamo ottenuto una stringa contenente il testo del file. A questo punto, separiamo le singole parole e le aggiungiamo alla lista di stringhe rappresentative del contenuto del file. Nella procedura vengono scartati i file con meno di 4 caratteri.

```
baseContent = content.Split(" ")
For Each x As String In baseContent
    If (x.Length > 3) Then
        processedContent.Add(x)
    End If
Next
```

Nel CD allegato sono presenti due provider per la lettura dei file che ci consentono di indicizzare dei file di tipo txt, rtf e doc.
Vediamo infine il main:

```
Imports System.IO
Imports System.Data.SqlClient
Imports System.Configuration
```

```
Module Program
  Private _StartDir As String = ""

  Sub Main(ByVal args() As String)

    Try

      [...]

      Dim dp As DirectoryParser = New
                                    DirectoryParser()

      [...]
      dp.FileParserClass =
        DataAccess.GetFileParserClass(Connection)
      [...]

      'Avvio lo scan della directory di partenza
      dp.StartParse(_StartDir)

    Catch ex As Exception
      Log.WriteLog("Errore: " & ex.Message)
    End Try

     [...]

  End Sub

End Module
```

Istanziamo un oggetto di tipo ***DirectoryParser*** e per esso recuperiamo le informazioni sui provider utilizzabili. Lanciamo quindi il metodo ricorsivo ***StartParse*** per iniziare a processare le directory del nostro HardDisk.

IL SISTEMA DI LOG

Anche il sistema di log dell'applicazione è realizzato utilizzando la Reflection.

Il metodo principale è ***WriteLog*** ed è quello che viene richiamato nei vari punti dell'applicazione quando si vuole scrivere una riga di log.

```
Public Shared Sub WriteLog(ByVal Text As String)

    Dim logMethod As String =
      ConfigurationManager.AppSettings("LogMethod")
    Try
        Dim Asm As Assembly =
                Assembly.GetExecutingAssembly()

        Dim t As Type =
                  Asm.GetType("FileIndexer.Log")
        Dim types() As Type = {GetType(String)}
        Dim param() As String = {Text}
        Dim m As MethodInfo =
                      t.GetMethod(logMethod, _
```

```
                  BindingFlags.NonPublic Or
                           BindingFlags.Static, _
                  Nothing, types, Nothing)
            m.Invoke(Nothing, param)

    Catch
            Throw New Exception("Gestore di Log
                                non implementato")
    End Try

End Sub
```

Il metodo consente di lanciare dinamicamente la funzione di log desiderata in base ad un parametro *logMethod* contenuto nel file ***App.config***.

CONCLUSIONI

La *Reflection* ci consente di realizzare applicazioni dinamiche e modulari; di scrivere del codice elegante e semplice a fronte di problemi a prima vista complessi. È comunque buona norma moderarne l'utilizzo in quanto esso produce un *overhead* in termini di prestazioni. Operazioni quali caricare un'*Assembly* a runtime, leggerne dinamicamente i tipi o invocarne i metodi hanno un costo in termini di tempo e vanno pertanto limitate. Una grande opportunità ma maneggiamola con cautela!

Carmelo Scuderi

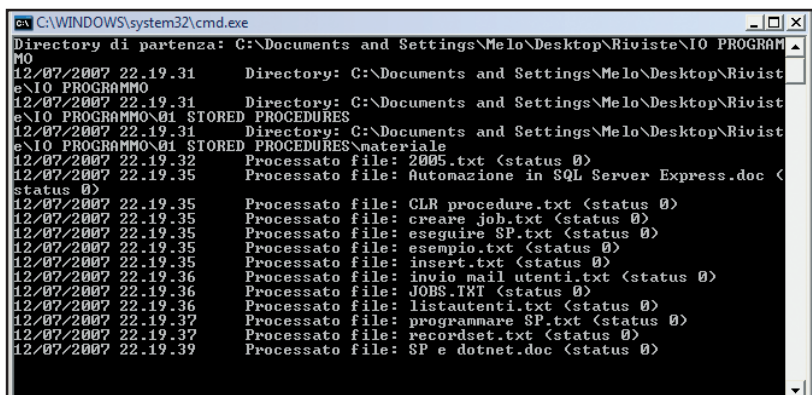



Figura 5: ***Il programma in esecuzione***

Carmelo Scuderi è ingegnere informatico. Si occupa di sviluppo software in ambiente .Net per una società di informatica di Milano. Gestisce un sito ricco di script e manuali per chi si affaccia al mondo della programmazione web (www.morpheusweb.it).

SYSTEM.TYPE

La classe System.Type sta alla base della Reflection, essa rappresenta un tipo del CLR e tramite essa è possibile ottenere le informazioni sui tipi. E' astratta e pertanto non può essere istanziata. In VB un oggetto di tipo Type viene generalmente ottenuto tramite il metodo GetType, in C# abbiamo l'equivalente typeof. Ulteriori informazioni all'indirizzo http://msdn2.microsoft.com/it-it/library/system.type(VS.80).aspx

UML E I DIAGRAMMI DI ATTIVITÀ

QUALCUNO DI VOI AVRÀ SICURAMENTE AVUTO A CHE FARE CON I CARI VECCHI "DIAGRAMMI DI FLUSSO". SI TRATTA DI UN MODO PER RAPPRESENTARE GRAFICAMENTE L'ALGORITMO DI UN SOFTWARE. UML CONTIENE IN SE OGGETTI SIMILI. VEDIAMO QUALI

Nei numeri precedenti abbiamo introdotto due importanti diagrammi UML: gli use case servono per rappresentare i requisiti utente; i diagrammi di classe invece sono utilizzati per rappresentare la struttura statica di parti del sistema.

In questo appuntamento descriveremo i diagrammi di attività. Questi possono essere considerati l'evoluzione dei cari e vecchi flow-chart (diagrammi di flusso). Con i diagrammi di attività è possibile descrivere una generica sequenza di operazioni. Questa può essere composta da una serie di chiamate a metodi delle classi del sistema. Oppure possono essere operazioni ad alto livello operate da un qualsiasi processo produttivo oppure organizzativo. Una delle caratteristiche dei diagrammi di attività è infatti quello di potersi adattare facilmente a diversi livelli di approfondimento o contesti. Rappresentano infatti uno strumento che può utilizzare l'analista di processo per descrivere il funzionamento ad alto livello di una organizzazione. Per esempio, un analista finanziario potrebbe realizzare una serie di diagrammi per descrivere la sequenza di operazioni che una banca intraprende quando gli viene richiesta l'erogazione di un finanziamento. Oppure, un responsabile di produzione in una industria alimentare potrebbe utilizzare una serie di diagrammi di attività per descrivere il funzionamento di una ipotetica linea di produzione. In questi casi i diagrammi non contengono attività svolte da un sistema informatico. Almeno non solo: potrebbero includere elementi elettronici, ma sicuramente includono persone, aziende, altri processi e così via. L'analista software può invece scendere in un dettaglio tecnico maggiore, legato al funzionamento di un sistema software. Per esempio, può utilizzare questi diagrammi per descrivere la sequenza di operazioni che sono necessarie per svolgere un determinato compito. Nell'ambito UML, questo compito potrebbe essere rappresentato benissimo da uno Use Case. Per esempio, si ipotizzi che l'analista debba descrivere la sequenza di operazioni necessarie perché l'impiegato di sportello di una banca possa registrare ed eseguire un bonifico richiesto da un cliente. Ovviamente questo tipo di processo include anche persone, come il cliente e l'impiegato di sportello. Ma indubbiamente questa sequenza di operazioni includerà la ricerca anagrafica del cliente nel database centrale dell'istituto di credito, l'individuazione del corretto rapporto o conto corrente, la verifica della disponibilità contabile per l'importo richiesto e così via. Per come sono stati elencati qui, queste funzionalità sono descritte ad alto livello e in modo concettuale. Questo fa parte ancora del piano di lavoro concettuale dei diagrammi di attività. Ma si potrebbero descrivere le stesse funzioni in termini di classi e invocazioni a metodo. Per esempio, la verifica della disponibilità dei fondi per eseguire l'operazione potrebbe essere scritta come:

```
importoRichiesto < conto.getSaldo()
```

In questo caso la prospettiva in cui si descrive il processo passa prettamente a un livello software. Scendendo ancora, è possibile ipotizzare un utilizzo molto dettagliato di questa tipologia di diagrammi, per esempio per descrivere algoritmi specifici. Questo ulteriore modo di impiegare i diagrammi di attività è più vicino al programmatore, il cui raggio di azione è solitamente limitato alla realizzazione vera e propria del software. Ovviamente, tutte le figure professionali possono giovare di queste diverse modalità di utilizzo dei diagrammi. Per esempio, il programmatore potrebbe essere in grado di comprendere maggiormente il sistema osservando i diagrammi di funzionamento ad alto livello. Potrebbe non essere vero il contrario: l'analista funzionale potrebbe non capire affatto il livello tecnico in quanto non sa programmare. A ogni modo, questo è un aspetto più organizzativo che specifico dello standard UML.

UTILIZZARE I DIAGRAMMI DI ATTIVITÀ

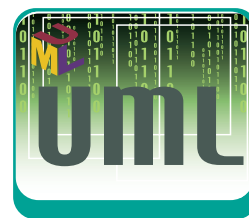
In ArgoUML (http://argouml.tigris.org/), è possibile disegnare diagrammi di attività selezionando la fun-

zione New Activity Diagram presente sotto il menu Create. Il pannello di disegno, posizionato nell'interfaccia utente di questo strumento UML in alto a destra, si predispone per disegnare. Si può notare il fatto che si sta creando un diagramma di attività osservando il primo pannello a sinistra. All'atto della creazione del diagramma è infatti comparso un nuovo nodo "Unnamed ActivityGraph" e sotto di questo un nuovo modello chiamato "untitledModel activity 0". Un altro elemento che palesa il fatto che si sta operando su un diagramma di attività è la barra degli strumenti presente in alto nel pannello di disegno.

Questa barra contiene icone per i comandi supportati nello specifico diagramma. La sua conformazione cambia in funzione del diagramma che si sta lavorando. Se infatti si fa clic sul pannello di sinistra e si seleziona un diagramma di tipo diverso, come per esempio un diagramma di classe, si vedrà che la barra degli strumenti riporta un contenuto diverso.

Fig. 1: Barra dei comandi dei diagrammi di attività

Nel caso dei diagrammi di sequenza i comandi disponibili sono (Figura 1):

- **freccia (select)**. Consente di selezionare uno o più elementi. Solitamente si sceglie anche per deselezionare uno dei simboli grafici presenti nel diagramma.
- **Broom**. Questo comando, una volta trascinato sull'area di disegno, permette di spostare orizzontalmente o verticalmente i simboli in esso presenti.
- **Attività (New Action State)**. Il riquadro con contorni arrotondati è il simbolo standard previsto da UML per rappresentare le singole attività presenti nel diagramma. All'interno di questo riquadro viene scritta una descrizione o riportato del codice che descrive quanto viene svolto in quel punto del flusso di esecuzione.
- **Freccia (New Transition)**. La freccia aperta identifica una transizione da una attività all'altra e permette di costruire l'ordine di esecuzione delle diverse attività semplicemente seguendo il flusso nel verso descritto dalla freccia.
- **Inizio (New Initial)**. Ogni diagramma di attività è caratterizzato dalla presenza di uno ed un solo stato iniziale, che è rappresentato da un pallino pieno. Questo simbolo identifica il punto di inizio del diagramma di attività e da esso parte la prima transizione che porta alla prima attività prevista dal flusso di lavoro.
- **Fine (New Final State)**. Per ogni inizio c'è anche una fine, che nei diagrammi di attività UML è rappresentata da un pallino pieno racchiuso in un cerchio. Questo identifica lo stato finale, il termine delle attività previste dal diagramma. Si noti che lo stato finale non identifica la fine dell'esecuzione del programma o l'uscita da una determinata funzione dello stesso, ma solo il completamento della sequenza di attività descritte nel diagramma. Il software, come i processi in generale, possono essere molto complessi. Potrebbero quindi essere necessari numerosi diagrammi per descrivere tutte le attività che sono coinvolte nel loro funzionamento.
- **Salto decisionale (New Junction)**. Il simbolo del rombo identifica nei diagrammi di attività un controllo decisionale. Per esempio, prima è stato descritto un esempio relativo alla disposizione di un bonifico bancario. È stato detto che uno dei controlli prevede la verifica della disponibilità di fondi. Questo controllo può essere rappresentato come salto decisionale. Si noti che da questo simbolo possono uscire più transizioni, ognuna relativa a un possibile risultato del controllo. Nel nostro caso potrebbero uscire due transizioni, una che porta al ramo di attività legate alla presenza di fondi, l'altra a quella che nega l'esecuzione dell'operazione. Si noti che, per contro, da un riquadro di attività può uscire solo una transizione.
- **New Fork e New Join**. I diagrammi delle attività consentono anche di specificare processi concorrenti. Questi si rappresentano come Fork e Join. Il primo di questi simboli consente di suddividere una transizione in due o più transizioni figlie. Lo scopo di Join è invece quello di riunire più transizioni in una singola. A livello software, quando si incontra un nuovo Fork significa che vengono creati più processi o thread paralleli ciascuno dei quali porta avanti autonomamente una sequenza di attività. Quando i diversi percorsi paralleli si riuniscono, è presente un Join.
- **Stato di chiamata (New Call State)**. Questo tipo speciale di attività identifica una invocazione a metodo.
- **Oggetto (New Object Flow State)**. Alcune volte si rende necessario esplicitare gli oggetti che vengono utilizzati per comunicare informazioni da uno stato all'altro.
- **Eventi**. Questo simbolo è una casella di selezione che consente di scegliere tra diversi tipi di eventi (chiamata, evento di modifica, segnale e segnale temporizzato). Questa funzione, come le due che seguono, si applicano a una specifica transizione. Per inserirle è dunque necessario selezionare l'icona relativa alla funzione e poi fare clic su una transizione.
- **Guardia (New Guard)**. Le guardie sono condizioni che definiscono se una certa condizione può avere luogo. Le guardie sono utilizzate anche per

NOTA

QUALE VERSIONE?

Argo UML, lo strumento utilizzato per la creazione dei diagrammi in questo articolo, supporta la versione 1.4 di UML. Quella più recente è la 2.0, che introduce diverse modifiche, ma per la quale non sono disponibili molti strumenti gratuiti.

contenere le condizioni relative a un salto decisionale.

- **Effetto**. Sotto questa casella di selezione sono presenti diverse opzioni (New Call Action, New Create Action, New Destroy Action, New Return Action, New Send Action, New Terminate Action, New Uninterpreted Action, New Action Sequence). L'effetto è la funzione che viene svolta dalla specifica attività. Per esempio, selezionando New Call Action si potrebbe indicare quale metodo di quale classe invocare per ottenere la funzionalità relativa all'attività. Questo potrebbe servire in fase di generazione automatica del codice.
- **Nota (New Comment)**. Consente di inserire un commento nel diagramma, funzionalità non peculiare dei diagrammi di sequenza ma in generale disponibile per tutte le tipologie di diagrammi previsti da questo standard.
- **Collegamento nota (New Comment Link)**. Le note sono collegate al contesto a cui fanno riferimento attraverso una linea tratteggiata che è attivabile con questa funzione.
- **Elemento grafico**. L'ultima casella di selezione della barra degli strumenti contiene una serie di elementi grafici standard ma non specifici di UML come il quadrato, il cerchio, l'etichetta, la linea continua, la spezzata, il poligono, la linea e il quadrato con bordi arrotondati. Attraverso questo menu è quindi possibile inserire nel disegno ulteriori elementi grafici scelti a piacere in questo elenco.

DISEGNARE UN DIAGRAMMA DI ATTIVITÀ

Per disegnare un diagramma di attività è necessario iniziare con uno stato iniziale. Per inserirlo nel piano di disegno è sufficiente cliccare il relativo simbolo sulla barra degli strumenti e poi in un punto qualsiasi dell'area di disegno. Argo UML disegna quindi lo stato iniziale. Posizionando il puntatore del mouse su questo elemento vengono presentati dei riquadri attivi. In particolare, vengono presentate due frecce, una a destra e una sotto allo stato iniziale. Per creare una nuova attività collegata allo stato iniziale è sufficiente fare clic su una di queste frecce. Argo UML creerà automaticamente una nuova attività e la posizionerà nel punto del diagramma che ritiene più opportuno. La stessa funzionalità è presente su tutti i simboli del diagramma, inclusi i nuovi riquadri di attività creati in questo modo.
Ovviamente è possibile trascinare qualsiasi elemento grafico del diagramma in punti diversi dell'area di disegno. Le transizioni presenti da un elemento all'altro vengono naturalmente manutenute, anche se la disposizione delle relative frecce seguono una logica poco gradevole esteticamente.

NOTA

Se interessa approfondire UML e il suo rapporto con i Design Pattern è possibile consultare il testo: Massimiliano Bigatti, "UML e Design Pattern. Notazioni grafiche e soluzioni per la progettazione", Hoepli Editore, ISBN: 978-88-203-3657-8

È possibile utilizzare questi meccanismi per produrre il numero di attività desiderate. Argo UML le rappresenta però come riquadri completamente bianchi, che è necessario riempire con una descrizione dell'attività. Per fare questo è sufficiente fare doppio clic sul riquadro dell'attività e digitare un testo descrittivo.
Per concludere questo primo diagramma è necessario inserire uno stato finale. Fare quindi clic sulla relativa icona e poi fare clic nel punto appropriato nell'area di disegno. Argo UML inserisce lo stato finale, che però è completamente staccato dal resto del diagramma. Per unirlo all'ultima attività che era stata inserita fare clic sull'icona Freccia nella barra degli strumenti e poi tracciare una linea dall'attività allo stato finale.
Per vedere i diagrammi di attività in pratica riprendiamo l'esempio illustrato nelle precedenti puntate. Nei numeri scorsi è stato infatti utilizzato UML per progettare un software per la gestione di una squadra calcistica: Team2007. Riprendiamo quel dominio funzionale disegnando un diagramma delle attività necessarie al tesseramento di un nuovo giocatore. Questo è illustrato in Figura 2. Sono presenti i seguenti elementi:

- stato iniziale;
- accesso all'elenco giocatori;
- selezione della funzione desiderata;
- inserimento dati giocatore;
- salvataggio delle informazioni;
- stato finale.

Fig. 2: Un esempio di semplice diagramma di attività.

UTILIZZO DEI SALTI CONDIZIONALI

In Figura 3 è presente un altro esempio di diagramma di attività. Questa volta è relativo alle operazioni svolte dall'allenatore per la definizione della squadra. Il processo è impostato in modo iterativo. Viene valutato un giocatore alla volta e, nel caso la valutazione sia positiva, viene inserito nella lista ufficiale. Come già spiegato, le biforcazioni di processo sono rappresentate da rombi, da cui possono uscire diverse transizioni. Su ciascuna di queste viene indicata la guardia, una descrizione che indica la condizione che deve essere soddisfatta perché il flusso di lavoro segua quella direzione.

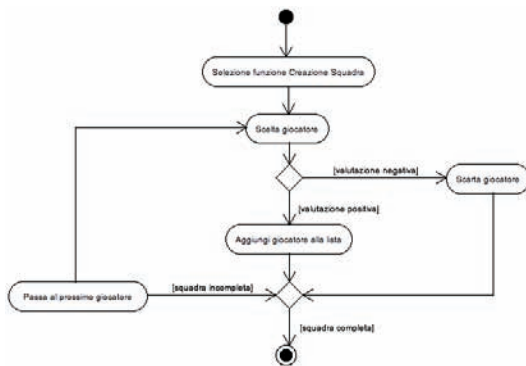



Fig. 3: Un diagramma di attività con salti condizionali.

Per realizzare concretamente questo diagramma utilizzando Argo UML è possibile procedere in sostanza come prima descritto, anche se ci sono un paio di particolari che è necessario considerare per inserire le guardie e per creare linee di transizione che seguono una linea non diretta.
L'inserimento dei salti condizionali avviene in modo simile al nodo finale: è necessario fare clic sull'icona relativa nella barra degli strumenti e poi fare clic su un punto dell'area di disegno. Per unire l'attività e il rombo è possibile procedere come già descritto, trascinando una nuova linea di transizione oppure posizionando il puntatore del mouse sull'attività, facendo clic e trascinando una delle icone che appaiono sui lati della stessa direttamente sul rombo. In questo modo si crea una transizione dall'attività al salto.
Una volta inserite le attività che seguono i diversi flussi di processo è necessario specificare le guardie che contengono le diverse condizioni. Per fare questo è sufficiente:

- fare clic sulla transizione su cui inserire la guardia;
- fare clic sull'icona della guardia nella barra degli strumenti;
- il pannello inferiore destro di Argo UML si modifica e si attiva la scheda Properties (Figura 4). All'interno di questa sono presenti tutti i campi necessari a compilare la guardia. L'elemento essenziale è il campo Expression. In cui è necessario digitare la descrizione della condizione.

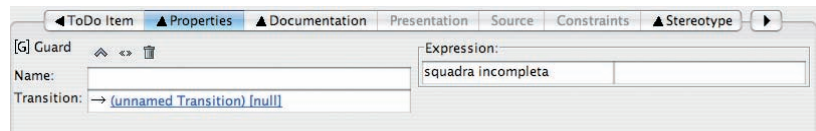
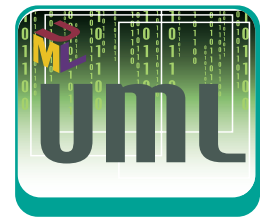



Fig. 4: Proprietà di una guardia.

Una volta inserita, la guardia viene visualizzata tra parentesi quadre direttamente sulla linea della transizione. Per modificarla è possibile ripetere le operazioni descritte. Questa volta nel pannello delle proprietà saranno però presenti le informazioni digitate in precedenza. In alternativa, è possibile procedere in uno dei due modi seguenti:

1. facendo direttamente doppio clic sul testo della guardia e digitando le modifiche in loco. Ovviamente in questo caso non è possibile accedere a tutte le informazioni, presenti invece nel pannello delle proprietà.

2. Selezionando la transizione da modificare e facendo doppio clic sul contenuto del campo Guard presente nel pannello Properties. In questo caso Argo UML salta direttamente alle proprietà della guardia.

Un ultimo particolare in merito a questo diagramma: per tracciare linee di transizione non dirette, come quella che unisce l'attività "Passa al prossimo giocatore" a "Scelta giocatore" è necessario utilizzare la freccia di transizione della barra degli strumenti e operando come descritto in precedenza, ma avendo l'accortezza di rilasciare il pulsante del mouse in un punto qualsiasi del piano di disegno. La linea di transizione non si interrompe, ma viene creato un angolo. Per creare ulteriori cambi di direzione sarà poi necessario fare un clic singolo con il mouse. Perché la transizione termini è necessario finire su un elemento grafico valido del diagramma.
Questa ulteriore possibilità offertaci da UML consente di otimizzare ulteriormente la fase di descrizione/progettazione del software

Massimiliano Bigatti

NOTA

LIBRI CONSIGLIATI

Un libro decisamente economico ma che rappresenta un buon reference di tutti gli elementi di UML 2.0 è: Enrico Amedeo "UML Pocket", Editore Apogeo, ISBN: 978-88-503-2627-3.

VB.NET PER MOBILE ELEMENTI VARIABILI

PROGRAMMARE UN'APPLICAZIONE PER POCKET PC È DIVENTATO ESTREMAMENTE SEMPLICE GRAZIE AGLI STRUMENTI MESSI A DISPOSIZIONE DA VISUAL BASIC.NET. REALIZZIAMO UN'APPLICAZIONE GRAFICA E INTRODUCIAMO LE STRUTTURE DI BASE...

Ai lettori già dalla prima puntata, non sarà sfuggita la versatilità e la potenza di VB.NET, che consente di creare qualsiasi applicazione per Pocket PC o Smartphone senza dover ricorrere a costose e, spesso, incomplete soluzioni proposte da terze parti. Per essere in grado di programmare questi veri e propri status symbol è necessario, però, partire dall'inizio, analizzando passo dopo passo tutti quei comandi e quelle istruzioni che vi consentiranno di avere tutti gli strumenti fondamentali sulla "punta delle vostre dite" (Bill Gates docet).

Fig. 1: L'applicazione d'esempio visualizzata all'interno di uno Smartphone

REQUISITI

Conoscenze richieste
Conoscenze di base sulla programmazione

Software
Microsoft Visual Studio.NET 2005, Windows Mobile 5.0 Pocket PC SDK e Windows Mobile 5.0 Smartphone SDK

Impegno

Tempo di realizzazione

LE VARIABILI

Com'è facile intuire, un programma lavora con dati di tipo diverso, cioè essenzialmente stringhe (ovvero sequenze di caratteri) e numeri; questi ultimi, poi, si dividono in numeri interi, decimali, che indichino valute, ecc. Questa distinzione è molto importante, perché ogni tipo di dato ha un'occupazione in memoria diversa: ad esempio, un numero intero occupa meno memoria di un numero decimale a precisione doppia.
In VB.NET, come nella maggior parte dei linguaggi di programmazione, le variabili sono utilizzate per memorizzare quei valori che si vogliono ricordare durante l'esecuzione del programma. Ad una variabile sono associati:

1. un nome, che rappresenta la parola utilizzata per riferirsi al valore che la variabile contiene
2. un tipo di dati, che determina il tipo di dati (numerico, data, carattere, ecc.) che la variabile può memorizzare.

Per dichiarare una variabile si utilizza l'istruzione Dim. La posizione e il contenuto della *Dim* consentono di determinare le caratteristiche di una variabile. Se, infatti, la variabile è dichiarata con Dim all'inizio del programma (subito prima del primo evento) è visibile in qualsiasi punto del programma. Se è dichiarata all'interno di un evento o procedura è visibile solo al loro interno. Vediamo un esempio.
Avviamo VB.NET, facciamo clic sul menu *File-New Project*, selezionate *Smart Device*, poi *Windows Mobile 5.0 Pocket PC* e fate doppio clic su *Device Application*. Visualizzate la Toolbox e fate doppio clic sull'oggetto Button, per riportarlo all'interno della Form1. Infine fate doppio clic sul pulsante stesso e digitate il software indicato di seguito.

```
Dim Contatore As Integer = 1
Dim Nome As String = "Enrico"
```

```
Dim Interruttore As Boolean = True

If Contatore = 1 Then
    MsgBox(Nome)
    Nome = "Chiara"
    Contatore += 1
End If
If Contatore = 2 Then
    MsgBox(Nome)
    Nome = "Gioia"
    Contatore += 1
End If
If Contatore = 3 Then
    MsgBox(Nome)
    Nome = "Sofia"
    Contatore += 1
End If
If Contatore = 4 Then
    MsgBox(Nome)
    Nome = "Milli"
    Contatore += 1
End If
If Contatore = 5 Then
    MsgBox(Nome)
    Interruttore = False
End If
If Not Interruttore Then
    MsgBox("ALGORITMO TERMINATO")
End If
```

Abbiamo, dunque, definito 3 variabili chiamate *Contatore, Nome* ed *Interruttore*. Si tratta di variabili locali, in quanto sono visibili solo all'interno dell'evento *Button1_Click()*. La clausola *As* che segue serve a definire il tipo dei dati che quella variabile ospiterá d'ora in poi e cioè rispettivamente: numerico (*Integer*), sequenza di caratteri (*String*) e booleano ossia che puó accettare solo due valori False o True (*Boolean*). Infine ogni variabile è stata inizializzata, ossia abbiamo definito un valore predefinito di partenza per ciascuna variabile: 1, "Enrico" e True.

Sfruttando l'istruzione *If*, che giá conoscete dalla scorsa puntata, abbiamo creato un piccolo programma che visualizza all'interno della Message Box una serie di nomi in sequenza. La spiegazione di questo algoritmo è semplice. Ogni *If* sulla variabile *Contatore* valuta qual è il valore corrente ed entra oppure no all'interno della If stessa; la *MsgBox()* visualizza il valore della variabile Nome ed attende che venga fatto clic sul pulsante OK; la variabile Nome viene progressivamente reimpostata con un altro nome; la variabile *Contatore* viene incrementata di uno ogni volta (+= 1); all'interno della penultima *If* viene modificata la variabile booleana in maniera tale che sia possibile accedere all'ultimo *If* per visualizzare il messaggio finale.

VETTORI E MATRICI

Quando parliamo di variabili non possiamo non fare riferimento alle matrici, che consentono di specificare una serie di variabili con lo stesso nome e di utilizzare un numero, detto indice, per distinguerle. In numerose situazioni ciò permette di creare codice più breve e più semplice, poiché è possibile impostare cicli in grado di gestire in modo efficiente qualsiasi numero di elementi facendo riferimento al

Tipo Visual Basic	Struttura dei tipi in Common Language Runtime	Allocazione di memoria nominale	Intervallo di valori
Boolean	**System.Boolean**	2 byte	**True** o **False.**
Byte	**System.Byte**	1 byte	Da 0 a 255 (senza segno).
Char	**System.Char**	2 byte	Da 0 a 65535 (senza segno).
Date	**System.DateTime**	8 byte	Dalle 0.00.00 dell'1 gennaio 0001 alle 23.59.59 del 31 dicembre 9999.
Decimal	**System.Decimal**	16 byte	Da 0 a +/-79.228.162.514.264.337.593.543.950.335 senza virgola; da 0 a +/-7,9228162514264337593543950335 con 28 decimali; il numero minore diverso da zero è +/-0,0000000000000000000000000001 (+/-1E-28).
Double (virgola mobile a precisione doppia)	**System.Double**	8 byte	Da -1,79769313486231570E+308 a -4,94065645841246544E-324 per valori negativi; da 4,94065645841246544E-324 a 1,79769313486231570E+308 per valori positivi.
Integer	**System.Int32**	4 byte	Da -2.147.483.648 a 2.147.483.647.
Long (long integer)	**System.Int64**	8 byte	Da –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.
Object	**System.Object** (classe)	4 byte	In una variabile di tipo **Object** è possibile memorizzare qualsiasi tipo.
Short	**System.Int16**	2 byte	Da -32.768 a 32.767.
Single (virgola mobile a precisione singola)	**System.Single**	4 byte	Da -3,402823E+38 a -1,401298E-45 per valori negativi; da 1,401298E-45 a 3,402823E+38 per valori positivi.
String (lunghezza variabile)	**System.String** (classe)	Dipende dalla piattaforma di implementazione	Da 0 a circa 2 miliardi di caratteri Unicode.
Tipo definito dall'utente (struttura)	(con caratteristiche ereditate da **System.ValueType**)	Dipende dalla piattaforma di implementazione	Ciascun membro della struttura presenta un proprio intervallo determinato dal tipo di dati specifico e indipendente dagli intervalli degli altri membri.

Fig. 2: Riepilogo dei tipi di dati di Visual Basic.NET.

INTERRUZIONE E COMBINAZIONE D'ISTRUZIONI NEL CODICE

Durante la scrittura del codice, vengono talvolta create istruzioni molto lunghe che richiedono uno scorrimento orizzontale nell'editor di codice e ne diminuiscono la leggibilitá. In questi casi, è opportuno suddividere l'istruzione su più righe utilizzando uno spazio seguito da un segno di sottolineatura. L'esempio che segue è equivalente a quanto indicato in precedenza.

```
Private Sub Button2_Click(ByVal
    sender As System.Object, _
ByVal e As
    System.EventArgs) Handles
        Button2.Click
...
End Sub
```

In altri casi, può essere opportuno riunire diverse istruzioni su un'unica riga, ad esempio quando sono state create numerose istruzioni brevi e si desidera ridurre lo spazio occupato. Il simbolo da utilizzare sono i due punti. Attenzione, però, a non abusarne, per evitare di rendere il programma poco leggibile. Anche qui l'esempio che segue è equivalente a quanto indicato in precedenza.

```
If Contatore = 1 Then
    MsgBox(Nome) : Nome =
        "Chiara" : Contatore += 1
End If
```

loro indice.
Le matrici ad una dimensione sono detti vettori o array unidimensionali. Ecco come dichiarare un vettore e come inzializzarne i valori.

```
Dim Qualifica(4) As String
Qualifica(0) = "Stagista"
Qualifica(1) = "Consulente"
Qualifica(2) = "Operaio"
Qualifica(3) = "Impiegato"
Qualifica(4) = "Dirigente"
```

In VB.NET il primo indice di vettori e matrici é sempre zero, come risulta chiaramente nell'esempio precedente.

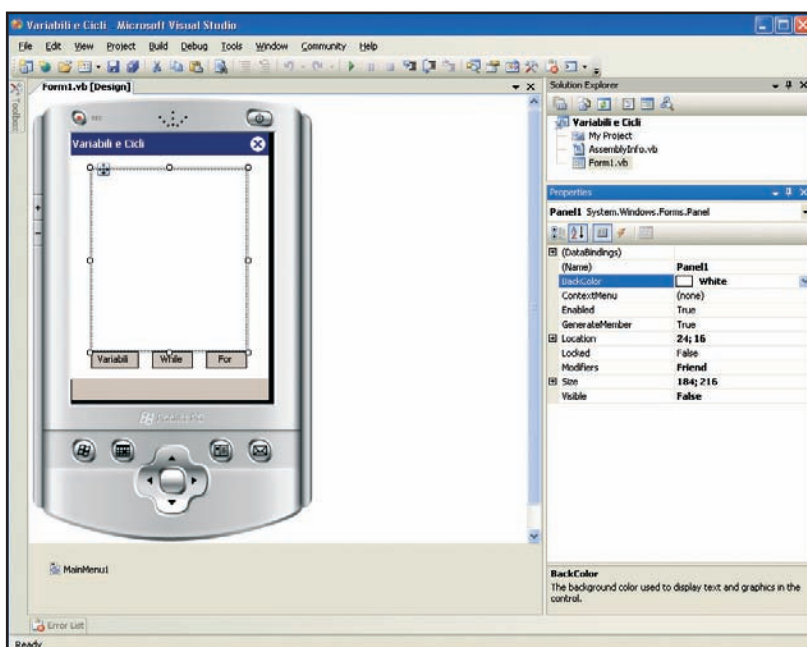



Fig. 3: L'applicazione d'esempio a design time.

Le matrici multidimensionali ammettono fino a 32 dimensioni. Il codice che segue, ad esempio, definisce una matrice bidimensionale di 5 righe (da 0 a 4) per 2 colonne (da 0 a 1).

```
Dim Citta_Provincia(4, 1) As String
```

```
Citta_Provincia(0, 0) = "Genzano di Roma" :
    Citta_Provincia(0, 1) = "RM"
Citta_Provincia(1, 0) = "Pianoro" :
    Citta_Provincia(1, 1) = "BO"
Citta_Provincia(2, 0) = "Comacchio" :
    Citta_Provincia(2, 1) = "FE"
Citta_Provincia(3, 0) = "Sassuolo" :
    Citta_Provincia(3, 1) = "MO"
Citta_Provincia(4, 0) = "Campoleone" :
    Citta_Provincia(4, 1) = "LT"
```

CAMBIAMENTI RISPETTO AL PASSATO

Visual Basic 6.0 è stata per anni la versione preferita dagli sviluppatori di Basic visuale. Ancora oggi esiste una foltissima schiera di programmatori VB6 che, sebbene non fornisca strumenti per smart device, vanta milioni di righe di codice che alimentano applicazioni sviluppate in tutto il mondo. Questo corso, al fine di coinvolgere anche i programmatori VB6, cercherà, quando se ne presenta l'occasione, di chiarire alcuni punti facendo un parallelismo con Visual Basic 6.0.
Uno dei cambiamenti più evidenti in Visual Basic.NET, rispetto alle versioni precedenti, riguarda i tipi di dato. Tale modifica si è resa necessaria per rendere i tipi di dato di VB.NET formalmente uguali a quelli utilizzati in C# e in C++ (gli altri due linguaggi principali di Visual Studio.NET). Il tipo di dato *Long* è stato sostituito da *Integer*, che quindi ora è usato per la rappresentazione di numeri interi a 32 bit; il vecchio Integer di VB6 e delle versioni precedenti, usato per i numeri a 16 bit, è diventato *Short*. In VB.NET il tipo Long serve per rappresentare i numeri a 64 bit. Comunque, per dichiarare questi tipi di dato, è possibile utilizzare anche i nomi con cui sono identificati

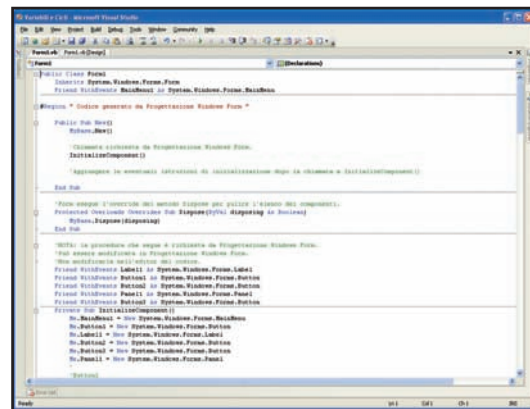



Fig. 4: Il codice generato da Progettazione Windows Form contiene tutte quelle istruzioni che VB.NET genera automaticamente quando viene creata la parte grafica. Non dovrebbero essere modificate manualmente..

COSTANTI

Un caso particolare di variabili sono le costanti. Esse mantengono sempre lo stesso valore e non possono essere modificate dopo la loro inizializzazione. Se si cerca di modificarne successivamente il valore VB.NET segnala subito in fase di disegno che "una costante non puó essere destinazione di un'assegnazione". La parola riservata Const prende il posto, in questo caso, di Dim.

```
Const PIgreco As Single = 3.14
Const Titolo As String =
    "Calcolatrice"
```

all'interno del Framework.NET, ossia delle librerie di base di Visual Studio.NET: invece di Short si può usare *Int16*, invece di Integer *Int32* e al posto di Long *Int64*.
Visual Basic.NET introduce delle novità anche per quanto riguarda l'area di validità delle variabili: ora le variabili sono locali rispetto alla porzione di codice in cui vengono definite. Se, ad esempio, si dichiara una variabile all'interno di un ciclo *For/Next*, essa non sarà più utilizzabile una volta usciti dal ciclo stesso. Se, ad esempio, si ha un algoritmo del tipo che segue:

```
Dim var As Short
For var = 0 To 20
    Dim ctrValore As Integer
    ...
Next
MsgBox (ctrValore)
```

Ancora una volta viene visualizzato un errore direttamente dall'editor del codice, in quanto la variabile *ctrValore* non è utilizzabile al di fuori del ciclo e, quindi, non risulta dichiarata in memoria.
Il tipo *Currency*, utilizzato in VB6 per i calcoli monetari e per calcoli a virgola fissa in cui la precisione rivestiva un'importanza particolare, è stato eliminato; al suo posto è possibile usare il nuovo tipo di dati *Decimal*, che ha lo stesso significato. Il tipo di dati Variant non esiste più: al suo posto è necessario usare il tipo *Object*, che comunque conserva tutte le caratteristiche del Variant, quindi può essere utilizzato per memorizzare qualsiasi tipo di dato primitivo, nonché *Empty, Null, Nothing*, oppure come puntatore ad un oggetto. Anche la gestione degli array è cambiata: ora gli array possono essere solo a base 0; non è quindi più possibile specificarne il limite inferiore.

Fig. 5: I colori casuali visualizzati in seguito al clic sul pulsante "While".

LE ISTRUZIONI ITERATIVE

I cicli sono forse le istruzioni piú importanti che abbiamo a disposizione. Il perché è intuibile: ogni applicazione viene eseguita in maniera iterativa e i cicli ci danno la possibilitá di minimizzare la scrittura del codice, creando algoritmi parametrizzati e facilmente manutenibili.
Il ciclo *While/End While* consente di eseguire un blocco di istruzioni per un numero indefinito di volte in base al valore *Boolean* di una condizione. Le istruzioni vengono ripetute fino a quando la condizione rimane *True*. Ad esempio il codice che segue visualizza un quadrato variopinto quando viene premuto il secondo pulsante, mentre lo nasconde al successivo clic sullo stesso pulsante. Questo perché il ciclo *While* viene terminato solo al secondo clic, quando la variabile *Ciclo* diventa di nuovo uguale a *False*.

```
    Private Sub Button2_Click(...) Handles
                                   Button1.Click
        Label1.Visible = False
```

Fig. 6: L'editor di VB.NET visualizza una casella gialla in cui avverte che la variabile in questione non è stata dichiarata.

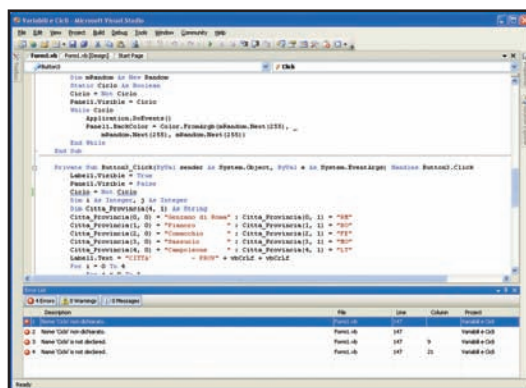



Fig. 7: La mancata dichiarazione di una variabile fa anche parte della Error List del compilatore.

```
        Dim mRandom As New Random
        Static Ciclo As Boolean
        Ciclo = Not Ciclo
        Panel1.Visible = Ciclo
        While Ciclo
            Application.DoEvents()
            Panel1.BackColor =
                Color.FromArgb(mRandom.Next(255), _
                mRandom.Next(255),
                                        mRandom.Next(255))
        End While
    End Sub
```

Nell'algoritmo precedente osservate in particolare *Ciclo = Not Ciclo*, che tradotto significa: se il valore booleano *Ciclo* è *False* impostalo a *True*, mentre se è uguale a *True*, impostalo a *False* (di conseguenza ad ogni pressione di *Button2*, *Ciclo* cambia valore). Si tratta di un esempio d'istruzione compatta che persegue lo scopo di minimizzare il codice presente nel programma a discapito, tuttavia, della chiarezza (e quindi manutenibilità) del software stesso. Per esplicitarlo in termini più leggibili si potrebbe scrivere quanto segue:

```
Static Ciclo As Boolean
If Ciclo = False Then
    Ciclo = True
Else
    Ciclo = False
End If
```

È importante notare che qui è stata utilizzata per la prima volta una variabile *Static* che mantiene il suo valore per tutta la durata di esecuzione del programma, ma solo nella procedura dove è stata definita (all'interno dell'evento *Button2_Click()*). Questo significa che la variabile *Ciclo* non è accessibile al di fuori dell'evento che la definisce e, un eventuale puntamento ad esso in un'altra procedura dell'applicazione comporterebbe un errore segnalato prontamente dall'editor di Visual Basic, ma evidenziato anche nella Error List che segue il tentativo di build di un progetto che ha degli errori (fare clic su *Build* e poi sul primo elemento visualizzato nel menu a comparsa). Il grosso vantaggio delle variabile statiche è che possono essere utilizzate come dei contenitori permanenti di valori, nei limiti, ovviamente, della durata di esecuzione in memoria dell'applicazione. Le altre tipologie di variabili non hanno questa caratteristica, a meno che non si utilizzino quelle di tipo *Public*, che però occupano più spazio in memoria in quanto sono sempre visibili ed accessibili, in qualsiasi funzione, procedura od evento del programma.
Continuando ad esaminare il codice della piccola applicazione appena presentata, diamo un'occhiata al tipo *Random*, ossia al cosiddetto generatore di numeri casuali. In questo caso è stata utilizzata per visualizzare colori a caso all'interno di *Panel1*, con l'ausilio della *FromArgb*, che è una classe predefinita di Visual Basic.NET. L'effetto che si produce è simile a quello di una TV che non riceve il segnale.
Infine si noti la *Application.DoEvents()* che serve a non dare priorità massima al ciclo While; se non ci fosse non potremmo piú interagire con il programma in fase di esecuzione e saremmo costretti a ricorrere al Task Manager di Windows per interromperlo. In effetti il metodo *DoEvents()* trasferisce il controllo al sistema operativo, che lo conserverà fino a quando non avrà terminato di elaborare gli eventi della propria coda. Se alcune parti del codice tengono occupato il processore per un tempo eccessivo, è sempre consigliabile utilizzare *Application.DoEvents()* ad intervalli regolari per consentire l'intervento di Windows, soprattutto perché eventi quali input da tastiera e operazioni con il mouse possano essere elaborati in tempi accettabili.
Un'altra istruzione iterativa, il *ciclo For/Next*, viene analizzata nel paragrafo che segue.

UTILIZZARE IL SOFTWARE ALLEGATO

Per avviare il progetto entrare nella cartella Variabili e Cicli e fare doppio clic sul file Variabili e Cicli.sln. Al fine di generare il codice eseguibile fare clic sul menu di primo livello Build e poi sull'elemento Build Variabili e Cicli. Il file .EXE è presente nel percorso \Cartella_CodiceAllegato\Variabili e Cicli\bin\Release e può essere copiato immediatamente all'interno dello Smartdevice tramite MS ActiveSync. Per comodità, in questo caso, la build del progetto è già stata effettuata.

IL CODICE DELL'ULTIMO PULSANTE

Innanzitutto dovete rendere invisibile *Panel1*, che qui non serve e visualizzare la *Label1*, all'interno della quale dovrete fare il display delle informazioni elaborate. Vi servono, poi, due variabili numeriche definite come *Short* che consentono di gestire i due cicli descritti di seguito. Ricordatevi d'impostare la proprietà *Font* di *Label1* con un font non proporzionale come Courier New, perché questo vi consentirà di visualizzare il contenuto della label con caratteri ben incolonnati come se si trattasse di una tabella formattata in precedenza.
Il ciclo *For/Next* consente di valutare una sequenza di espressioni più volte. Si distingue, pertanto, dall'espressione *If* nella quale il programma valuta ogni espressione una sola volta. Questa istruzione iterativa è consigliabile quando si conosce in anticipo il numero di volte in cui le espressioni devono essere valutate.
Dal codice contenuto nell'ultimo pulsante *Button3_Click()* estrapoliamo il doppio ciclo *For/Next* che permette di scorrere righe e colonne della matrice inizializzata in precedenza Citta_Provincia() e di visualizzarne in una label i valori. Da osservare la costante *vbCrLf* utilizzata due volte, che impone al testo una doppia andata a capo (corrisponde, infatti, ad un doppio Invio).

Fig. 8: Il font del controllo Label1 non deve essere proporzionale.

```
Private Sub Button3_Click(...) Handles
                                        Button3.Click
    Panel1.Visible = False
    Label1.Visible = True
    Dim i As Short, j As Short
    Dim Citta_Provincia(4, 1) As String
    Citta_Provincia(0, 0) = "Genzano di Roma" :
                          Citta_Provincia(0, 1) = "RM"
    Citta_Provincia(1, 0) = "Pianoro          " :
                          Citta_Provincia(1, 1) = "BO"
    Citta_Provincia(2, 0) = "Comacchio       " :

                          0Citta_Provincia(2, 1) = "FE"
    Citta_Provincia(3, 0) = "Sassuolo        " :
                          Citta_Provincia(3, 1) = "MO"

    Citta_Provincia(4, 0) = "Campoleone      " :
                          Citta_Provincia(4, 1) = "LT"
    Label1.Text = "CITTA'           - PROV" +
                                     vbCrLf + vbCrLf
    For i = 0 To 4
        For j = 0 To 1

            Label1.Text = Label1.Text +
                                     Citta_Provincia
                                                (i, j)
            If j = 0 Then
                Label1.Text = Label1.Text + " - "
            Else
                Label1.Text = Label1.Text + vbCrLf
            End If
        Next
    Next
End Sub
```

Enrico Bottari

Fig. 9: La visualizzazione della tabella delle città e delle province dopo che è stato fatto clic sul pulsante "For".

I COMPONENTI VISUALI IN JSF

ANALIZZIAMO IL MODELLO A COMPONENTI DI JSF E VEDIAMO COME SIA POSSIBILE SFRUTTARE AL MASSIMO QUESTA CARATTERISTICA PER AUMENTARE LA RIUTILIZZABILITÀ. INOLTRE INTRODURREMO I COMPONENTI DI BASE DELL'ARCHITETTURA

Dopo i primi articoli su JSF, dove abbiamo incominciato a prendere confidenza con questo framework, ora avremo modo di analizzare uno dei suoi punti di forza: il modello a componenti.

IL MODELLO A COMPONENTI

All'interno del framework JSF possiamo trovare diverse tipologie di componenti, visuali e non. Le componenti visuali sono quelle che permettono allo sviluppatore di manipolare le informazioni che verranno visualizzate nell'interfaccia grafica della web application. Le componenti non visuali sono invece quelle relative alla validazione, conversione dei dati. In JSF esiste una gerarchia di componenti visuali, che permette allo sviluppatore di definire un'interfaccia web e di creare altri componenti non visuali (ad esempio dei managed bean) che possono gestire il ciclo di vita di questi componenti. A capo di questa gerarchia troviamo la classe UIComponent, quindi tutti i componenti che utilizzeremo nella nostra interfaccia web (customizzati o non) derivano da questa classe. La pagina web deve essere considerata come una vista (View) all'interno della quale possiamo inserire diversi componenti. In questo caso stiamo parlando di componenti che vengono renderizzati in HTML (ad esempio HtmlInputText), ma nessuno ci vieta di ragionare in termini più astratti, perché ogni componente può essere renderizzato in altri modi. Infatti JSF permette di gestire i componenti, associando ad essi diversi renderer. Ad esempio se volessimo portare una nostra applicazione web sui terminali mobili (wml/xhtml) potremmo semplicemente associare un renderer diverso ai componenti che abbiamo utilizzato. Per comprendere come una pagina JSF sia una composizione di componenti, possiamo vedere il seguente codice

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="f"
          uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h"
          uri="http://java.sun.com/jsf/html"%>

<html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Gerarchia</title>
  </head>
  <body>
  <f:view>
    <h:form id="loginForm">
        <h:inputText size="30"/><br>
        <h:message
          for="loginForm">Ciao</h:message>
        <h:commandButton value="Submit"
                                        /><br>
    </h:form>

    <h:dataTable value="#{helpBean.lista}"
                                  var="item" >
    ....
    ....
    ....
    </h:dataTable>
  </f:view>
  </body>
</html>
```

Questa è una semplice pagina JSF con qualche componente al suo interno (non scendiamo ora nei dettagli). Questa pagina può essere rappresentata con il diagramma qui a fianco. Possiamo vedere che come elemento radice della nostra pagina abbiamo una UIViewRoot, ovvero una vista. Al suo interno sono presenti diversi componenti, che possiamo utilizzare per comporre la nostra pagina.

In questo caso abbiamo tralasciato i sotto-componenti relativi a HtmlDataTable (che vedremo in un paragrafo successivo), ma la cosa importante è capire appunto che tutti gli elementi che andiamo a posizionare all'interno di una pagina web possono essere gestiti e renderizzati in maniera molto semplice con JSF.

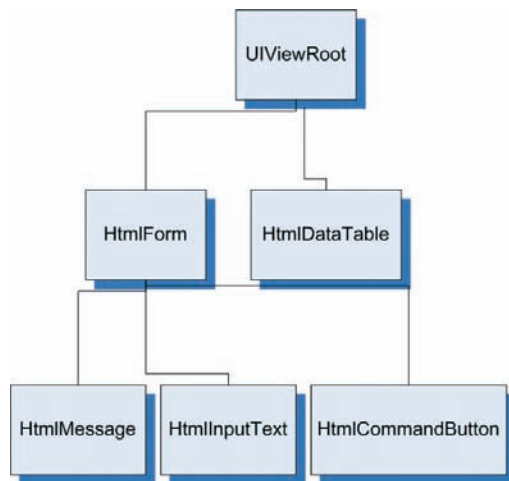



***Fig. 1:** Gerarchia dei componenti presenti all'interno di una possibile pagina JSF*

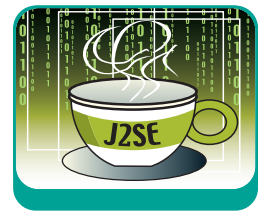
Abbiamo detto che tutti i componenti visuali in JSF derivano da una classe base, UIComponent. A questo punto dobbiamo suddividere i componenti in due categorie, quelli che generano un'azione (come un bottone, un link) che implementano l'interfaccia ActionSource2 e quelli che gestiscono l'input utente (caselle di testo, scelte multiple), che implementano l'interfaccia ValueHolder o EditableValueHolder. Vedremo ora come gestire i diversi tipi di componenti con un bean.

GESTIONE DEI COMPONENTI

Quello che vogliamo vedere ora è la gestione dei componenti visuali che inseriamo all'interno della pagina attraverso un bean. Prima di tutto quindi dobbiamo realizzare un bean e inserirlo nel file di configurazione di JSF. Qui di seguito potete vedere l'implementazione del bean che andremo ad utilizzare

```
package testjsf;

import javax.faces.component.html.*;

public class HelpBean {

   private HtmlInputText inputText1;
   private HtmlOutputText outputText1;

   public HelpBean() {
   }

   public HtmlInputText getInputText1() {
      return inputText1;
   }
   public void setInputText1(HtmlInputText
                                  inputText1) {
      this.inputText1 = inputText1;
   }

   public HtmlOutputText getOutputText1() {
      return outputText1;
   }

   public void setOutputText1(HtmlOutputText
                                  outputText1) {
      this.outputText1 = outputText1;
   }

   public String scambio() {
      outputText1.setValue(inputText1.getValue());
      inputText1.setValue(new String("Informazione
                                  inserita"));
      inputText1.setReadonly(true);
      return "success";
   }
}
```

Cerchiamo ora di capire cosa possiamo fare con questo bean. Al suo interno sono presenti due variabili che rappresentano due componenti che visualizzeremo all'interno della pagina web. La prima è un'istanza di HtmlInputText, ovvero un semplice campo di testo, mentre la seconda un'istanza di HtmlOutputText, un testo HTML. Oltre a queste due variabili e i rispettivi metodi get/set, abbiamo un metodo scambio(), che prende il valore contenuto nel campo di input, lo inserisce in quello di output, cambia il testo dell'input e lo setta come readonly. In questo modo stiamo manovrando i componenti presenti nella nostra pagina, attraverso questo bean, come se fossimo dietro le quinte. Andiamo ora a definire il bean all'interno del file faces-config.xml, che si trova nella directory WEB-INF del nostro progetto

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces
                                 Config 1.1//EN"
```

```
  "http://java.sun.com/dtd/web-
                    facesconfig_1_1.dtd">

<faces-config>

  <managed-bean>
    <managed-bean-name>helpBean</managed-
                              bean-name>
    <managed-bean-
    class>testjsf.HelpBean</managed-bean-class>
    <managed-bean-scope>request</managed-
                              bean-scope>
  </managed-bean>

</faces-config>
```

Grazie alla entry che è stata riportata nel file di configurazione, le nostre pagine web possono utilizzare un bean, usando il nome helpBean, con scope request. Quest'ultima cosa significa che le informazioni saranno contenute nel bean solo per una semplice request. Se ad esempio volevamo inserire informazioni che potevano servire per tutta la durata della web application (un carrello della spesa ad esempio), lo scope doveva essere almeno session. Vediamo ora come è possibile collegare i componenti che inseriamo in una pagina al bean che abbiamo appena visto

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@taglib prefix="f"
            uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h"
            uri="http://java.sun.com/jsf/html"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
                          Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type"
            content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>

    <f:view>
      <h1><h:outputText value="JavaServer
                          Faces" /></h1>
      <br>
      <h:form>
        <h:inputText
      binding="#{helpBean.inputText1}" /><br>
        <h:commandButton value="Scambia"
          action="#{helpBean.scambio}" /><br>
        <h:outputText value="In attesa di
            input.." binding="#{helpBean.outputText1}" />
        </h:form>
        <br>
      </f:view>

    </body>
</html>
```

All'interno della pagina abbiamo inserito i due componenti di input ed output, un bottone ed una form. Come potete vedere i componenti di input ed output riportano nel tag di definizione la proprietà binding definita. Questo attributo ci permette di collegare il componente ad una proprietà definita in un bean. Nel bottone invece abbiamo definito l'attributo action, inserendo il metodo che ci permette di eseguire il codice che volevamo. Come abbiamo già visto nei precedenti articoli, quando inseriamo un metodo collegato ad un bottone di un form, di solito vogliamo inviare delle informazioni, cambiare pagina o qualcosa del genere. Per fare ciò dobbiamo definire l'outcome, il risultato che restituisce il metodo invocato, nel file di navigazione di JSF e decidere cosa fare in base a questo risultato. In questo caso noi vogliamo semplicemente rimanere nella stessa pagina e visualizzare i cambiamenti che si verificano. Infatti se andiamo a testare questa pagina, vedremo che succede quello che volevamo, ovvero il testo nell'input passa nell'output e il campo dell'input diventa readonly.

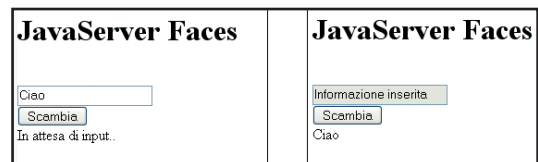



Fig. 2: Risultato della manipolazione dei componenti

ALTRI COMPONENTI VISUALI

Passiamo ora ad un esempio più interessante, la gestione di un componente che ci permette di visualizzare delle tabelle HTML, HtmlDataTable. Questo deriva da UIData, che è un componente che permette di gestire collezioni di informazioni. In questo caso le informazioni vengono disposte in una tabella HTML. Questo componente è presente all'interno della distribuzione JSF, quindi possiamo subito vedere come funziona senza scaricare altri file. La prima cosa che ci serve è una collezione di informazioni da visualizzare. Diciamo che vogliamo gestire una lista di film,

con titolo, categoria e giudizio. Prima di tutto dobbiamo creare un semplice classe che mappa queste informazioni

```
package testjsf;

public class Film {

    private String titolo;
    private String categoria;
    private String giudizio;

    public Film() {
    }

    public String getTitolo() {
        return titolo;
    }

    public void setTitolo(String titolo) {
        this.titolo = titolo;
    }

    public String getCategoria() {
        return categoria;
    }

    public void setCategoria(String categoria) {
        this.categoria = categoria;
    }

    public String getGiudizio() {
        return giudizio;
    }

    public void setGiudizio(String giudizio) {
        this.giudizio = giudizio;
    }

}
```

Ora dobbiamo creare un metodo in un bean che ci permetta di trasferire queste informazioni nella pagina. In questo caso utilizzeremo lo stesso bean che abbiamo utilizzato precedentemente, HelpBean, senza fare troppo caso all'organizzazione del codice.

```
public Film[] getFilms() {
        Film film1=new Film();
        film1.setTitolo("Lady in the water");
        film1.setCategoria("Fantasia/Thriller");
        film1.setGiudizio("Bellissimo");
        Film film2=new Film();
        film2.setTitolo("Transformer");
        film2.setCategoria("Azione/Fantascienza");
        film2.setGiudizio("Tanta tanta azione");
        Film film3=new Film();
```

```
        film3.setTitolo("The order");
        film3.setCategoria("Horror/Thriller");
        film3.setGiudizio("Bleah!");
        Film[] films=new Film[3];
        films[0]=film1;
        films[1]=film2;
        films[2]=film3;
        return films;
}
```

Questo metodo non fa altro che creare 3 oggetti Film ed inserirli in un array che viene restituito. Per visualizzare la nostra tabella, non dobbiamo far altro che inserire il tag relativo in un pagina e richiamare questo metodo per popolarla

```
<h1><h:outputText value="La
                tabella"/></h1>
<h:dataTable value="#{helpBean.films}"
                var="item" >
    <f:facet name="header">
        <h:outputText value="I film che ho
                visto recentemente" />
    </f:facet>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Titolo" />
        </f:facet>
        <h:outputText value="#{item.titolo}"
                />
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Categoria"
                />
        </f:facet>
        <h:outputText
            value="#{item.categoria}" />
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Giudizio" />
        </f:facet>
        <h:outputText
            value="#{item.giudizio}"/>
    </h:column>
    <f:facet name="footer">
        <h:outputText value="Copyright
                2007" />
    </f:facet>
</h:dataTable>
```

Il tag h:dataTable ha bisogno di una collezione di oggetti per poter poi rappresentare queste informazioni come una tabella. In questo caso abbiamo collegato il suo attributo value al metodo films() che abbiamo predentemen-

NOTA

QUALCHE LIBRO SU JSF

Java Server Faces – La guida completa, McGraw-Hill
http://www.informatica.mcgrawhill.it/esperto/java_server_faces.asp

Java Server Faces in Action, Manning
http://www.manning.com/mann/

te visto. Dobbiamo poi definire il nome di una variabile che ci permette di scorrere tutta la collezione ed utilizzare i vari elementi, come quando in un foreach utilizziamo una variabile temporanea. Dopo di ciò abbiamo definito 3 diverse colonne attraverso il tag h:column, dove andiamo ad inserire l'attributo che deve essere riportato. Come potete vedere, sopra e sotto alle 3 colonne ci sono altri due tag, header e footer, che vengono tradotti in HTML con i tag thead e tfoot, utili per inserire informazioni sopra e sotto la tabella.

La tabella

I film che ho visto recentemente		
Titolo	**Categoria**	**Giudizio**
Lady in the water	Fantasia/Thriller	Bellissimo
Transformer	Azione/Fantascienza	Tanta tanta azione
The order	Horror/Thriller	Bleah!
Copyright 2007		

Fig. 3: La tabella risultante da HtmlDataTable a cui è stato applicato un CSS

Una delle cose interessanti che troviamo se sviluppiamo una web application in JSF è che esiste la possibilità di realizzare dei propri componenti, che magari contengono al loro interno semplici componenti base ma disposti in una certa maniera con un certo funzionamento. Se vogliamo riutilizzare componenti definiti da altre persone, la prima cosa che dobbiamo fare vedere il progetto MyFaces Tomahawk (http://myfaces.apache.org/tomahawk/index.html), che definisce diversi componenti preconfezionati da utilizzare. Tanto per fare un esempio, all'interno di questo progetto troviamo il componente InputHtml, che è praticamente un componente di input che ci permette di avere in una pagina web un classico editor HTML WYSIWYG (What You See Is What You Get).

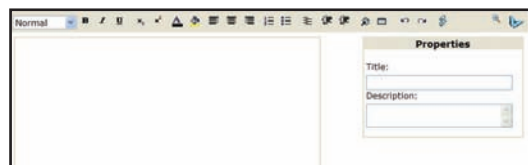



Fig. 4: Editor HTML di Tomahawk

Per inserire questo editor non dobbiamo far altro che mettere nella nostra web application il seguente codice (selezionando i vari attributi che si vogliono abilitare)

```
<t:inputHtml value="String"
                style="CSSClass"
                fallback="{true|false}"
                type="Constant"
                allowExternalLinks="{true|false}"
                addKupuLogo="{true|false}"
                showAllToolBoxes="{true|false}"
                allowEditSource="{true|false}"
                showPropertiesToolBox="{true|false}"
                showLinksToolBox="{true|false}"
                showImagesToolBox="{true|false}"
                showTablesToolBox="{true|false}"
                showDebugToolBox="{true|false}"
showCleanupExpressionsToolBox="{true|false}"/>
```

Se vogliamo utilizzare questo componente possiamo inserire direttamente il jar di Tomahawk (e le due dipendenze) nella nostra web application ed utilizzare i suoi tag, anche se non utilizziamo MyFaces. Ciò nonostante consiglio di utilizzare l'implementazione di Apache nel caso in cui si voglia ricorrere a questa libreria, per non dover andare incontro a possibili problemi di API.

UN LISTENER SUL COMPONENTE

Vediamo ora un altro aspetto importante dei componenti visuali di JSF, ovvero la possibilità di intercettare eventi sui vari componenti. In questo caso daremo solo un accenno a questo interessante tema che sarà trattato in futuri articoli, ciò nonostante era necessario per avere una visione completa sul modello a componenti che Java Server Faces fornisce con le sue API. Abbiamo visto che possiamo associare le proprietà di un bean alle componenti visuali della nostra interfaccia grafica. Come nello sviluppo di GUI, ogni componente può scatenare degli eventi, in base ai quali

AMBIENTI DI PROGRAMMAZIONE

Esistono diversi ambienti visuali dove è possibile creare un progetto JSF e realizzare pagine, componenti, navigazione in maniera grafica. Chiaramente l'utilizzo di questi prodotti è consigliato solo quando c'è già almeno una base di conoscenza del framework, perchè altrimenti anche piccole modifiche che l'interfaccia non permette possono risultare impossibili e/o complicate.

NetBeans + Visual Web Pack
http://www.netbeans.org/
Sun Java Studio Creator
http://developers.sun.com/jscreator/
Exadel Studio
http://www.exadel.com/
Oracle JDeveloper 10g
http://www.oracle.com/technology/software/products/jdev/index.html

può essere implementata una certa logica di business. Ad esempio abbiamo già visto negli articoli precedenti come legare dei componenti visuali ad un certo validatore, che può essere uno standard definito in JSF oppure uno implementato tramite una nostra classe o metodo di un bean. Anche in quel caso possiamo pensare ai controlli di validazione come a degli eventi che vengono scatenati dal componente. In questo caso vogliamo associare un listener ad un componente, per renderci conto quando il valore di questo viene modificato. Di seguito viene riportato il codice che serve per legare un listener ad un componente di input

```
<h:form>
    <h:inputText
        binding="#{helpBean.inputText2}"
  valueChangeListener="#{helpBean.exampleListener}
        " onchange="this.form.submit()" /><br>
    <h:inputText
        binding="#{helpBean.inputText3}" />
</h:form>
```

Abbiamo definito due componenti di input, li abbiamo legati a dei valori di un bean e su uno dei due componenti abbiamo settato l'attributo valueChangeListener. Questo campo ci permette di specificare chi deve essere notificato nel caso in cui sia scatenato l'evento ValueChangeEvent. In questo caso abbiamo utilizzato il nostro HelpBean, con un metodo che prende in input questo evento.

```
public void exampleListener(ValueChangeEvent vce) {
    if
      (((String)vce.getNewValue()).equals("pippo"))
        inputText3.setValue("Hai scritto pippo!");
    else
        inputText3.setValue("Avrai scritto qualche
                                        altra cosa");
}
```

Questo metodo d'esempio non fa altro che inserire una stringa nel secondo campo di input in base a quello che è stato scritto nel primo campo. Per far scatenare questo evento avremmo bisogno che l'utente invii la form dove sono contenuti questi due campi, ma in questo caso abbiamo inserito il campo onchange che praticamente utilizza JavaScript per rendersi conto della modifica del campo in tempo reale ed inviarla al server senza che ci sia bisogno da parte nostra dell'invio tramite bottone.
Questo è solo un assaggio di quello che è possibile fare utilizzando listener in JSF, però era utile vedere come fosse possibile utilizzare dei listener sui vari componenti visuali che andiamo a gestire.

CONCLUSIONI

Abbiamo visto come sia semplice gestire componenti visuali in JSF e come questo modello sia poi utilizzabile attraverso dei bean di gestione che catturano eventi e/o mappano informazioni. Una delle prime obiezioni che possono essere fatte riguarda la gestione ancora non troppo pulita all'interno della vista (JSP). I componenti visuali sono gestiti tramite taglib e quindi obbligano comunque ad una prolissità nel codice che viene riportato in queste pagine. Giustamente quando dobbiamo gestire interfacce complesse che vanno ben oltre i semplici componenti che abbiamo visto, la manutenzione delle pagine può diventare un compito pesante ed oneroso. Uno dei possibili metodi per risolvere questo problema è quello di utilizzare Facelets (https://facelets.dev.java.net/). Questa libreria ci permette di realizzare una semplice composizione di componenti, creando un componente "container", all'interno del quale possono essere definiti diversi componenti visuali che vogliamo trattare in maniera monolitica. Il modello a componenti consente da un lato la massima riutilizzabilità e dall'altro la massima estendibilità. Si tratta di un'architettura di base che costituisce un elemento chiave per tutti i linguaggi attuali, JSF non poteva fare eccezione.
Nei prossimi articoli di questo corso avremo modo di soffermarci sulla possibilità di costruire applicazione AJAX in JSF e vedere lo sviluppo completo di un applicazione d'esempio che possa riassumere alcune parti già trattate, per vedere come far interagire tutte le feature messe a disposizione da questo framework.

Federico Paparoni

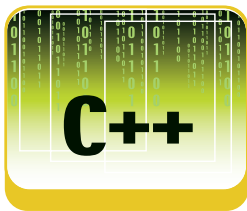
L'AUTORE

L'autore, Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

COMPONENTI DI TERZE PARTI

È possibile utilizzare ed integrare in maniera molto semplice componenti scritti da altre persone. La SUN ha pubblicato il progetto Woodstock (https://woodstock.dev.java.net**) che è praticamente un repository di componenti JSF opensource che possono essere riutilizzati. Un altro posto dove poter cercare componenti che possiamo utilizzare è JSFCentral (**http://www.jsfcentral.com/products/**), sito dove vengono pubblicati anche link a tutorial e notizie riguardanti il mondo JSF.**

METTI D'ACCORDO LINUX E WINDOWS

QUALÈ L'APPLICAZIONE MINIMA CHE POSSIAMO FARE CON LE WXWIDGETS? COME SI GESTISCONO I PULSANTI? E COME SONO STRUTTURATE LE FINESTRE? A QUESTI INTERROGATIVI E A MOLTI ALTRI RISPONDEREMO IN QUESTO ARTICOLO

Benvenuti al nostro secondo appuntamento con wxWidgets. Nello scorso numero abbiamo descritto la situazione attuale delle librerie GUI e illustrato come installare wxWidgets sul proprio sistema. Se vi siete persi la prima puntata, siete ancora in tempo per compilare la libreria seguendo le istruzioni sul Web, e saltare a bordo.
In questo numero illustreremo wxWidgets dalle fondamenta, senza dare nulla per scontato. Partiremo dalla gestione dell'applicazione, fino alla creazione di semplici finestre. Tutto questo ci permetterà di discutere una sfilza di dettagli senza i quali non si può comprendere il reale funzionamento di questo framework.

IL PROGRAMMA MINIMO

Partiamo con un quiz. Qual è il programma più piccolo che è possibile scrivere in C++? Pensateci un attimo… questa è la risposta:

```
int main() {}
```

Allo stesso modo, chiediamoci qual è il programma più piccolo che è possibile scrivere con wxWidgets. Eccolo:

```
#include "wx/wx.h"
IMPLEMENT_APP(wxApp)
```

Questo suggerisce un paio di considerazioni. Innanzitutto, che per usare wxWidgets dovrete sempre includere il file "wx/wx.h" che si occupa di includere tutte le sue componenti – a patto che abbiate impostato il vostro IDE come abbiamo visto nello scorso appuntamento.
Altra considerazione: wxWidgets trasforma radicalmente la struttura di un programma C++, tanto che il main – punto di ingresso fondamentale dell'applicazione, obbligatorio secondo lo standard – sparisce dalla vista del programmatore finale.
Al suo posto compare una macro spaventosa dal nome *IMPLEMENT_APP*. E qui si apre una discussione che abbiamo già accennato nello scorso numero, ma che è bene approfondire, anche per coloro che si fossero messi in ascolto solo in questo momento. WxWidgets è un'ottima libreria, probabilmente la migliore in questo momento. Ciò nonostante non usatela mai come guida di stile, perché – per una serie di fattori, fra i quali il principale è il fatto che è "vecchia" – molte scelte architetturali sono semplicemente orrende. Le macro, usate come mezzo per nascondere la complessità dell'applicazione, sono probabilmente uno degli ingredienti di wxWidgets che incontrerete più spesso – e che più vi rivolteranno, se siete dei veri amanti del C++ e della sicurezza a tempo di compilazione. In questo caso, *IMPLEMENT_APP* viene usata al posto del main, perché in alcune piattaforme la programmazione GUI prevede punti di ingresso alternativi (ad esempio in Windows si usa storicamente una funzione chiamata WinMain). *IMPLEMENT_APP* si prende cura di inizializzare la vostra applicazione correttamente, e di gestirla attraverso un oggetto globale di tipo **wxApp** – che è a tutti gli effetti una classe singleton. *WxApp* contiene eventuali dati globali (ad esempio: gli argomenti passati, in argc e argv) e gestisce il flusso dell'esecuzione e la coda dei messaggi. Se provate a lanciare il programma vedrete che compila senza problemi, ma forse rimarrete un po' delusi al momento dell'esecuzione.

DERIVARE DA WXAPP

L'"applicazione minima", infatti, continua a girare senza far niente e non c'è verso di fermarla – se non uccidendo il processo. Questo è il comportamento standard della classe *wxApp*. Dato che non è molto interessante, non userete mai wxApp direttamente, ma creerete una classe derivata nella quale ridefinirete alcune funzioni membro virtuali. In particolare, *OnInit* permette di prendere il controllo del flusso di esecuzione nel momento in cui l'applicazione verrà inizializzata. Detto in altre parole: in wxWidgets

wxApp::OnInit fa le veci del main.
Ecco un codice in cui deriviamo da *wxApp* per mostrare a schermo un simpatico messaggio "Hello World!":

```
#include "wx/wx.h"

class CiaoApp : public wxApp
{
public:
    bool OnInit() {
        wxMessageBox(_T("Ciao Mondo!"));
        return true;
    }
};

IMPLEMENT_APP(CiaoApp)
```

***Figura 1:** Il risultato dell'esecuzione di wxMessageBox, su Windows XP.*

L'output dell'esecuzione sotto Windows è visibile in figura 1. Anche stavolta abbiamo il nostro *IMPLEMENT_APP*, ma passiamo come argomento non più *wxApp*, bensì una sua erede: *CiaoApp*.
In *CiaoApp* ridefiniamo la funzione membro *OnInit*, al cui interno richiamiamo la funzione *wxMessageBox*. Questa costituisce il metodo più semplice per notificare qualcosa all'utente – e quindi è molto comoda per il debugging.
Come primo argomento, *wxMessageBox* richiede una stringa. Qui occorre aprire una parentesi. Quando usate delle stringhe in wxWidgets, è sempre meglio che le passiate alla macro *_T*, come vedete nell'esempio. *_T* cambia il testo a seconda della configurazione che avete scelto in sede di compilazione. Ad esempio, se avete seguito i consigli della volta scorsa compilando per Unicode, probabilmente "Ciao Mondo!" diventerà L"Ciao Mondo!". *_T* serve, appunto, a non pensarci, e a non dover modificare nulla nel caso decidiate in futuro di cambiare modalità.
Già che siamo in argomento stringhe: il primo parametro di *wxMessageBox* è, in realtà, di tipo *wxString*. Si tratta di un tipo che *wxWidgets* usa ogniqualvolta abbia a che fare con le stringhe (è un'altra testimonianza della vecchiaia della libreria, nata prima dei contenitori standard e quindi ignara di oggetti efficacissimi come *std::string*). Fortunatamente esistono molte conversioni automatiche da e verso *wxString*, pertanto raramente vi capiterà di averci a che fare direttamente.
Per finire, fate attenzione al comportamento dell'applicazione: quando cliccate su "*OK*" il messaggio sparisce, ma il programma continua a girare imperterrito, proprio come nell'esempio precedente! La "colpa" è del valore restituito dalla funzione. *OnInit*, infatti, restituisce un valore booeano: true se l'applicazione deve continuare a girare, e *false* se deve chiudersi immediatamente, subito dopo l'inizializzazione. Se cambiate la riga finale in:

```
return false;
```

l'applicazione si autoterminerà appena premerete il pulsante OK.

FRAME E FINESTRE

Per ora sappiamo solo creare applicazioni che continuano finché non gli si spara, oppure che terminano appena dopo l'inizializzazione. Per fare un passo avanti, dobbiamo cominciare a costruire delle finestre vere e proprie. In wxWidgets ci sono molti modi per costruire una finestra, ma uno dei più immediati è quello di istanziare un nuovo oggetto di tipo wxFrame, così:

```
#include "wx/wx.h"

class CiaoApp : public wxApp
{
public:
    bool OnInit() {
        wxFrame* frame = new wxFrame(0,
                        wxID_ANY, _T("Ciao, Mondo!"));
        frame->Show();
        return true;
    }
};
IMPLEMENT_APP(CiaoApp)
```

Come al solito, analizziamo prima il codice, e poi il comportamento dell'applicazione – visibile in **Figura 2**.
In OnInit, stavolta, abbiamo creato un nuovo oggetto di tipo wxFrame in memoria dinamica. Il costruttore di un wxFrame deve ricevere almeno

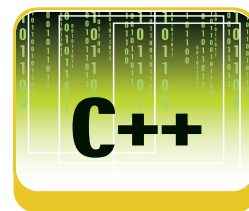
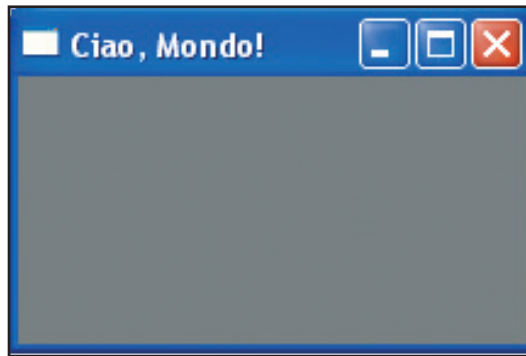
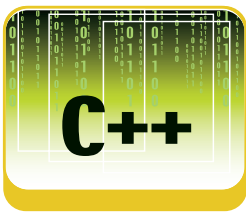



Figura 2: Il frame "Ciao Mondo", su Windows XP

tre argomenti:

- **Un puntatore alla finestra madre**. Nel nostro caso, il frame non ha genitori, pertanto passiamo il puntatore nullo (0). Grazie a questo parametro, è possibile creare delle vere e proprie gerarchie di finestre. Una finestra madre ha alcuni "diritti" sulle figlie – ad esempio, può rintracciarle a partire da un ID o sfogliando un elenco. Quando una finestra madre viene chiusa, tutte le eventuali finestre figlie vengono automaticamente chiuse.
- **Un IDentificatore**. Come accennato, una finestra madre può "chiamare" una figlia conoscendone il nome. Tale *nome* viene deciso al momento della costruzione proprio attraverso questo parametro. Quando non si prevede di utilizzare l'identificatore – come in questo caso –, si può usare l'ID predefinito ***wxID_ANY***.
- **Il titolo**. Un oggetto wxString (vedi precedente paragrafo), che indica il testo che dovrà comparire nella barra del titolo.

Notate che richiamiamo un operatore *new* senza il corrispondente *delete*. Questo è di solito un sintomo di memory leak, ma in questo caso siamo tranquilli: quando la finestra verrà chiusa l'oggetto di tipo *wxFrame* si autodistruggerà da solo.
A differenza di altri componenti di wxWidgets dal comportamento più disinvolto, i frame sono creature timide e non si mostrano a video a seguito della sola costruzione. Richiamando il metodo ***Show*** li rendiamo visibili all'utente. Nel caso in cui, in futuro, volessimo cambiare idea, potremmo sempre nasconderli con una chiamata a *Show(false)*.
Notate, infine, il valore di ritorno di *OnInit*: ***true***. Questo indica che il programma può partire, avviando il *ciclo degli eventi*. Vedremo più avanti cosa sono gli eventi; nel frattempo notate che l'applicazione non continua più all'infinito: termina non appena il frame viene chiuso. Questo è il comportamento di wxWidgets: quando l'ultima finestra è stata chiusa, il programma finisce.

OnInit, quindi, dovrebbe sempre restituire *true*, a meno che non succeda qualcosa di male durante l'inizializzazione che impedisca l'avvio del programma. Per assicurarsene una routine d'inizializzazione dovrebbe sempre prevedere una chiamata alla funzione membro *OnInit* esposta dalla classe base. Così:

```
#include "wx/wx.h"

class CiaoApp : public wxApp
{
public:
   bool OnInit() {
      //richiama l'inizializzazione "standard"
      if (!wxApp::OnInit())
         return false;

      //inizializziamo l'applicazione
      wxFrame* frame = new wxFrame(0,
                        wxID_ANY, _T("Ciao, Mondo!"));
      frame->Show();
      return true;
   }
};

IMPLEMENT_APP(CiaoApp)
```

Per motivi di spazio, d'ora in poi darò spesso per scontato che tutte le implementazioni di *OnInit* seguano questo modello.

AGGIUNGERE UN PULSANTE

Il frame che abbiamo creato in Figura 2 funziona, ma è spoglio e inutile. Per dargli un senso dobbiamo riempirlo di controlli (o, nel gergo di wxWidgets… widgets!), come caselle di testo, etichette, menu e, soprattutto, pulsanti!
Tutto questo è molto più semplice di quanto potrebbe sembrare, anzi: abbiamo appena visto come si fa nell'ultimo paragrafo. In wxWidgets, infatti, tutti i widget sono finestre a tutti gli effetti, nel senso che derivano dalla classe wxWindow, e quindi si costruiscono più o meno tutti come wxFrame.
Un pulsante, ad esempio, appartiene alla classe wxButton. Per inserirlo nel frame, è sufficiente costruirlo indicando il frame stesso come suo genitore, così:

```
//...
bool OnInit() {
   wxFrame* frame = new wxFrame(0,
                     wxID_ANY, _T("Ciao, Mondo!"));
   frame->Show();
```

```
    wxButton* pulsante = new wxButton(
        frame, wxID_ANY, _T("Ma a che serve 'sto
                                    programma?")
    );

    return true;
}
```

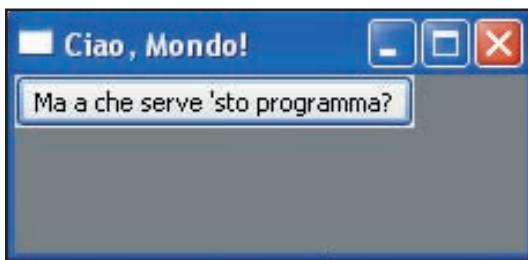



***Figura 3:** Al frame è stato aggiunto un pulsante*

In Figura 3 potete ammirare il nostro *frame* col suo nuovo pulsante. E potete vedere che questo viene posizionato in alto a sinistra (posizione [0,0]) e in una dimensione minima. Questo tipo di parametri possono essere aggiustati a piacimento grazie agli ulteriori argomenti del costruttore. Ad esempio, per posizionare il controllo in [20, 20] e dargli una dimensione di [200, 100] scriviamo:

```
wxButton* pulsante = new wxButton (
    frame, wxID_ANY, _T("Ma a che serve 'sto
                                programma?"),
    wxPoint(20,20), wxSize(200,100)
);
```

wxPoint e *wxSize* sono due strutture interne di *wxWidgets*, che si usano rispettivamente per indicare punti e dimensioni. Questo tipo di posizionamento viene detto **assoluto** e, come vedremo in uno dei prossimi appuntamenti, è piuttosto primitivo.
Ma torniamo al punto essenziale: il pulsante ci fa una domanda irriverente: "a che serve 'sto programma?". Vorremmo dire che serve a molto, a farci imparare come funziona *wxWidgets* partendo dalle fondamenta – ma per farlo dovremmo in qualche modo reagire alla pressione del pulsante. E come si fa? Finché non sapremo rispondere, purtroppo, avrà ragione lui: questo programma non serve a niente.

DERIVARE DA WXFRAME

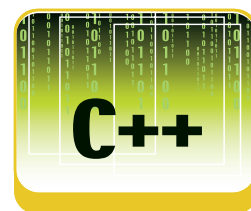
Prima di dare una bella lezione al pulsante, rivediamo un attimo il nostro approccio. Anche se siamo circondati da macro, infatti, dobbiamo pur sempre ricordarci che stiamo programmando in C++. A chi appartiene il pulsante? Al frame, non all'applicazione. Secondo il sempiterno Criterio di Parnas (leggi: incapsulamento), dovrebbe essere solo il frame ad avere accesso al pulsante e nessun altro.
Dobbiamo quindi ottenere un nuovo frame con un pulsante incapsulato, creando una classe derivata, così:

```
class NuovoFrame : public wxFrame
{
public:
  NuovoFrame() :
    wxFrame(0, wxID_ANY, _T("Ciao,
                              Mondo!") {
      wxButton* pulsante = new wxButton(
        this, wxID_ABOUT, _T("Ma a che
                  serve 'sto programma?")
      );
  }
};
```

Il nuovo tipo di frame (che chiamiamo, guarda un po', *NuovoFrame*) aggiungerà il pulsante al momento della sua costruzione. Nello scrivere un costruttore per la classe, evitiamo di chiedere argomenti inutili all'utente e passiamo direttamente i parametri che ci interessano al costruttore della classe base.
Notate che ora il genitore del pulsante è *this* (cioè il frame stesso), e che ho cambiato l'identificatore in *wxID_ABOUT*. Questo è uno dei tanti ID predefiniti di wxWidgets, e in generale si usa per indicare un componente che soddisfi la richiesta: "Ma a che serve 'sto programma?". Proprio il nostro caso.
Anche se non sembra, abbiamo fatto un bel passo avanti: ora è possibile costruire un frame nuovo con un pulsante preincapsulato che ha come identificatore wxID_ABOUT, semplicemente scrivendo questo:

```
NuovoFrame* frame = new NuovoFrame();
```

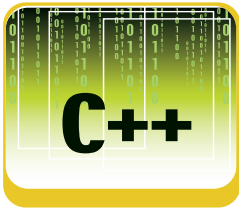
GESTIRE GLI EVENTI

E ora possiamo rispondere per le rime al simpatico pulsante. Per questo dobbiamo catturare l'evento che si genera quando questo viene premuto. WxWidgets, infatti, resta continuamente in ascolto degli eventi che vengono generati dal sistema operativo e li ridistribuisce ai vari oggetti secondo uno schema ben preciso, fatto di strutture chiamate ***event table***, che è fondamentale capir bene.

NOTA

BIBLIOGRAFIA

Fate riferimento alla documentazione ufficiale (http://wxwidgets.org/docs/) per avere informazioni su classi e funzioni della libreria.

Se, come me, preferite i libri di carta, Cross-Platform Gui Programming with wxWidgets, scritto da Julian Smart (lo stesso autore della libreria), è un buon punto di partenza e un primo punto di riferimento per il framework.

Infine, ricordate che wxWidgets è open source, e spesso il codice sorgente è la migliore delle documentazioni!

Una Event Table è in sostanza una grossa serie di macro (tanto per cambiare) associate ad un oggetto (come *NuovoFrame*) spesso derivato da una classe base (come *wxFrame*).
Lo scopo di una Event Table non è altro che associare un evento (ad esempio: "è stato premuto il pulsante wxID_ABOUT") ad una funzione, detta in gergo **handler** (ad esempio: *NuovoFrame::DaiInformazioni*). Quando scatta l'evento, viene richiamato l'handler.
NuovoFrame può cambiare così:

```
class NuovoFrame : public wxFrame
{
    //...Implementazione... (vedi sopra)

    //Handler
    void DaiInformazioni(wxCommandEvent&
                                              event) {
        wxMessageBox(_T("Serve eccome, stupido
                                          pulsante!"));
    }

    DECLARE_EVENT_TABLE()
};
```

L'implementazione è esattamente la stessa di prima, tuttavia abbiamo aggiunto due elementi:

- **L'handler *DaiInformazioni***: Come dicevamo, useremo questa funzione per rispondere alla pressione del pulsante. Gli handler sono sempre funzioni non virtuali, che non restituiscono alcun valore, e alle quali viene passato come argomento un *evento* – ovverosia una classe derivata da *wxEvent*, che contiene tutte le informazioni del caso (a eventi diversi corrispondono informazioni diverse, e pertanto tipi diversi. In questo caso abbiamo un evento del tipo "click su pulsante", e quindi un **wxCommandEvent**).
- **La macro *DECLARE_EVENT_TABLE***: Questa macro viene espansa in righe di codice che servono a far capire a wxWidgets che alla classe verrà associata in seguito una event table. Se vi dimenticate questa riga, il compilatore non capirà l'event table relativa.

Quindi possiamo passare alla scrittura dell'event table vera e propria:

```
BEGIN_EVENT_TABLE(NuovoFrame, wxFrame)
    EVT_BUTTON(wxID_ABOUT,
                              NuovoFrame::DaiInformazioni)
END_EVENT_TABLE()
```

Tutto qui. La tabella è delimitata dalle macro ***BEGIN_EVENT_TABLE(Classe, ClasseBase)*** e ***END_EVENT_TABLE***. L'unica voce della tabella è quella relativa al pulsante. ***EVT_BUTTON*** è una macro (sì, ancora un'altra) che indica un evento associato a un pulsante (che genera, come abbiamo anticipato, un *wxCommandEvent*). I due argomenti della macro sono: l'identificatore (quale pulsante viene premuto?), e l'handler (quale funzione bisogna richiamare se si verifica l'evento?).
Ora possiamo compilare il codice, ottenendo la nostra rivincita, immortalata in Figura 4.

***Figura 4:** La pressione del pulsante ha attivato l'handler DaiInformazioni.*

UN CONTROLLO "MUSICALE"

Comprendere le dinamiche del sistema degli eventi in wxWidgets è fondamentale. Nell'ultimo paragrafo abbiamo visto come costruire un'event table per una classe personalizzata. Qui cerchiamo di capire cosa succede dietro le scene.
WxWidgets è in continua attesa di nuovi eventi, che gli vengono segnalati dal Sistema Operativo. Quando questo accade, velocemente fa una ricerca su tutte le event table che riguardano l'oggetto relativo al messaggio, e – una volta che trova una corrispondenza – smette di cercarne altre.
Proprio per questo, l'ordine in cui vengono cercate le corrispondenze è fondamentale. WxWidgets parte dalla classe più derivata e via via risale fino alla classe base. È un metodo sensato, perché ci permette di sovraccaricare i comportamenti dei controlli, sicuri che le nostre modifiche verranno trovate *prima*, e quindi sostituiranno il comportamento standard.
Un esempio pratico sarà d'aiuto: proviamo a realizzare un controllo in cui ciò che ogni volta che l'utente preme un tasto venga visualizzato il carattere e contemporaneamente emesso un suono. Chiameremo il nostro controllo *MusicText* e lo costruiremo a partire da

wxTextCtrl. Questa è l'event table del nostro controllo; la macro ***EVT_KEY_DOWN*** è associata alla pressione di un tasto:

```
BEGIN_EVENT_TABLE(MusicText, wxTextCtrl)
    EVT_KEY_DOWN(MusicText::OnKeyDown)
END_EVENT_TABLE()
```

EVT_KEY_DOWN richiede un unico argomento: l'handler. Questo sarà la funzione *OnKeyDown* del nostro controllo *MusicText*. L'evento passato alla funzione è un ***wxKeyEvent***, che contiene tutte le informazioni relative al tasto premuto – che a noi ora non interessano. Per generare il suono, possiamo usare la funzione ***wxBell*** che richiama il suono "di allarme" predefinito:

```
//prima della Event Table, definiamo la classe
class MusicText : public wxTextCtrl
{
public:
    MusicText(wxWindow* parent_) :
                    wxTextCtrl(parent_, wxID_ANY) {}
    void OnKeyDown(wxKeyEvent& event) {
        wxBell();
    }
    DECLARE_EVENT_TABLE()
};
```

Se provate ad usare questo controllo, vi accorgerete che qualcosa non va: "suona", ma non scrive! Perché? Provate a pensarci, alla luce di quanto appena detto.
Ecco cosa succede: quando l'utente preme un tasto, il Sistema Operativo genera l'evento e wxWidgets lo prende in carica. L'oggetto che l'ha generato è di tipo *MusicText*, pertanto viene scansionata questa classe. WxWidgets cerca una corrispondenza e la trova in *EVT_KEY_DOWN*: l'utente ha premuto un tasto e il gestore bisogna richiamare *MusicText::OnKeyDown*. Fatto questo, wxWidgets si ferma, e non consegna l'evento alla classe base *wxTextCtrl*, di fatto scavalcandola. Ecco perché non viene visualizzato niente.
In casi come questo, bisogna segnalare a wxWidgets che si vuole che la ricerca nelle Event Table continui, richiamando la funzione ***Skip*** dell'evento passato al gestore come argomento. Così:

```
//...
void OnKeyDown(wxKeyEvent& event) {
    wxBell();
    event.Skip();
}
```

Alla fine della funzione, wxWidgets controllerà se *Skip* è stata richiamata. In questo caso non fermerà la ricerca nelle event table, ma la continuerà nella classe base immediatamente precedente, e così via. Infatti, ora il controllo funziona correttamente.
Grazie alle event table, quindi, si possono costruire sia finestre piene di controlli, sia controlli personalizzati pronti per il riutilizzo – nello stesso identico modo, rapido e preciso.

CONTROLLI E IDENTIFICATORI

A questo punto siete pronti per sperimentare liberamente e costruire le vostre interfacce grafiche. Probabilmente vi troverete a gestire decine di controlli su un singolo frame, e capiterà spesso che dobbiate farvi riferimento. Ci sono diversi metodi:

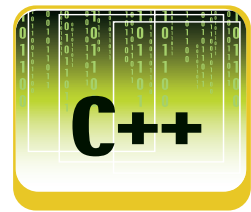
- **Memorizzate un puntatore** all'interno della vostra classe. Ad esempio:

```
class MioFrame : public wxFrame
{
public:
    wxButton* pulsante_;

    MioFrame(wxFrame* parent) :
      wxFrame(parent, wxID_ANY, _T("Mio
                                        Frame")) {
          pulsante_ = new wxButton(this,
                        wxID_ANY, _T("Mio Pulsante"));
    }
};
```

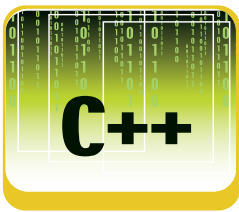
Qui abbiamo creato un Frame che contiene un pulsante, al quale fa direttamente riferimento tramite il membro puntatore *pulsante_*.
Questo è il metodo più semplice per permettere ad un *MioFrame*, in futuro, di accedere al proprio pulsante, ad esempio per cambiarne l'etichetta:

```
void MioFrame::CambiaEtichettaPulsante() {
    pulsante_->SetLabel(_T("Tuo Pulsante"));
}
```

- **Associate il controllo ad un identificatore:**

```
class MioFrame : public wxFrame
{
public:
    MioFrame(wxFrame* parent) :
      wxFrame(parent, wxID_ANY, _T("Mio
                                        Frame")) {
          new wxButton(this, wxID_APPLY, _T("Mio
                                        Pulsante"));
    }
```

DIALOGBLOCKS

Come avete visto, scrivere a mano il codice per frame e controlli è complicato e frustrante. Proprio per questo sono nati diversi IDE che permettono di svolgere il lavoro graficamente (alla Visual Basic), e ottenere del codice pronto da copincollare. Il mio preferito si chiama DialogBlocks.

```
};
```

Stavolta non abbiamo memorizzato alcun puntatore, ma abbiamo associato al pulsante un ID predefinito (*wxID_APPLY*, ma avremmo anche potuto usare una qualunque costante positiva impostata da noi). In *CambiaEtichettaPulsante*, ci basterà richiamare il metodo **GetWindowChild** per recuperare un puntatore al pulsante.

```
void MioFrame::CambiaEtichettaPulsante() {
    GetWindowChild(wxID_APPLY)-
                        >SetLabel(_T("Tuo Pulsante"));
}
```

Il secondo sistema è effettivamente più pratico e ci consente di risparmiare righe di codice – ma, ovviamente, esistono degli svantaggi. Innanzitutto, il ricorso alla funzione *GetWindowChild* implica una ricerca in una lista. In secondo luogo, il puntatore restituito è sempre di tipo ***wxWindow***, una classe base da cui ereditano tutti i controlli. Per accedere alle funzioni e ai membri del controllo bisogna ricorrere ad un downcasting – operazione notoriamente brutta e insicura.

```
wxButton* MioFrame::GetPulsante() {
    return
     static_cast<wxButton*>(GetWindowChild(wxID_A
                                          PPLY));
}
```

DALLA TEORIA ALLA PRATICA

Ora che abbiamo tutti questi elementi in mano, vediamo di realizzare la semplice applicazione visibile in Figura 5. Si tratta di un'interfaccia per un telefonino: l'utente preme i pulsanti e sul display in alto vengono scritti i simboli relativi. Se state usando quest'articolo come un tutorial, il mio consiglio è di provare a realizzare il programmino da soli, e poi di "sbirciare" se e quando sarete in difficoltà (se siete davvero pigri, potete anche copincollare il codice dal CD allegato). Qui di seguito presento il codice per intero, intervallato da commenti e spiegazioni.

```
#include "wx/wx.h"
```

Definiamo un nuovo Frame (*PhoneFrame*) che conterrà i pulsanti e il display.

```
class PhoneFrame : public wxFrame
{
public:
```

Ora il costruttore, che servirà a creare il display. Normalmente è una buona norma delegare il "lavoro sporco" della costuzione degli oggetti ad una funzione membro esterna al costruttore. Norma che qui non seguiremo, per ragioni di spazio.

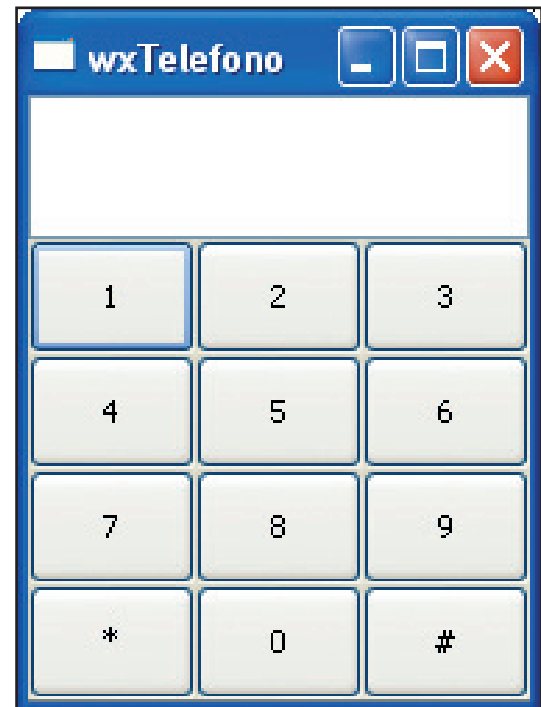



Figura 5: L'applicazione di esempio: wxTelefono

```
    PhoneFrame() :
      wxFrame(0, wxID_ANY, _T("wxTelefono"),
         wxPoint(0,0), wxSize(188, 245)) {
```

Notate la strana dimensione della finestra: 188x245. È tutto calcolato in maniera che il frame contenga esattamente i controlli. Questo tipo di disposizione, come potete immaginare, è molto tediosa da realizzare, quasi impossibile da manutenere, e ben poco cross-platform. Nei prossimi appuntamenti vedremo come usare elementi a dimensionamento dinamico, in maniera da risolvere il problema. Qui cerchiamo di salvare il possibile, usando delle costanti che rendano agevoli eventuali modifiche:

```
            //--- Crea il display ---
            const wxSize DisplaySize(180, 50);
```

```
display_ = new wxTextCtrl (
            this, wxID_ANY, _T(""),
            wxPoint(0,0), DisplaySize
          );
        //Lo rende impossibile da modificare
        display_->SetEditable(false);
```

Abbiamo appena creato il display. Niente di straordinario, se si esclude la chiamata a *SetEditable*, che è una delle tante funzioni membro di un *wxTextCtrl*. Ora creiamo i pulsanti. Per non usare dodici lunghissime istruzioni *new*, usiamo un ciclo:

```
    //--- Crea i pulsanti ---
    const wxSize ButtonSize(60, 40);
    const wxString label[4][3] = {
        {_T("1"), _T("2"), _T("3")},
        {_T("4"), _T("5"), _T("6")},
      {_T("7"), _T("8"), _T("9")},
      {_T("*"), _T("0"), _T("#")}
    };
    for (unsigned int y=0; y<4; ++y) {
        for (unsigned int x=0; x<3; ++x) {
            new wxButton (

                this, ID_BUTTON, label[y][x],
                    wxPoint (
                        x * ButtonSize.x,
                        DisplaySize.y + y *
                                        ButtonSize.y
                    ),
                    ButtonSize,
            );
        }
    }
  }
```

Fine del costruttore. Ora scriviamo una funzione membro (OnButton) che sarà richiamata ogni volta che verrà premuto un pulsante.

```
  void OnButton(wxCommandEvent& event) {
```

Poiché tutti i pulsanti hanno lo stesso ID (*ID_BUTTON*, che definiremo privatamente), tutti i pulsanti richiameranno questa funzione membro. Per sapere quale pulsante è stato premuto, richiamiamo la funzione membro *GetEventObject* dell'oggetto event. Questa restituisce il pulsante premuto, sotto forma di puntatore a *wxWindow*, che ritrasformeremo in un wxButton per mezzo di un downcast.

```
    wxButton* button =
  static_cast<wxButton*>(event.GetEventObject());
```

Ora possiamo aggiungere al display (tramite la funzione membro *AppendText*) l'etichetta del pulsante premuto (ottenuta tramite *GetLabel()*).

```
    display_->AppendText(button-
                              >GetLabel());
  }
private:
```

Privatamente definiamo *ID_BUTTON* (come 1, ma va bene qualsiasi intero positivo) e il puntatore display. Notate che ho combinato in questo esempio i due metodi esposti nel paragrafo precedente: per i pulsanti ho usato un ID, riuscendo così ad associarli tutti allo stesso evento; per il display, invece, è più comodo memorizzare un puntatore diretto.

```
    static const int ID_BUTTON  = 1;
    wxTextCtrl* display_;
```

E infine non dimentichiamoci dell'event table. Internamente alla classe la dichiariamo

```
    DECLARE_EVENT_TABLE()
};
```

ed esternamente la definiamo, associando i messaggi dei pulsanti con identificatore *ID_BUTTON* alla funzione *OnButton*.

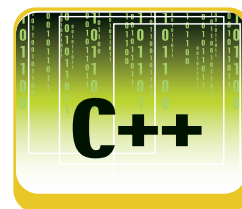
```
BEGIN_EVENT_TABLE(PhoneFrame, wxFrame)
  EVT_BUTTON(PhoneFrame::ID_BUTTON,
                      PhoneFrame::OnButton)
END_EVENT_TABLE()
```

Ora rimane soltanto l'applicazione, che mostrerà a video un *PhoneFrame* e terminerà alla sua chiusura:

```
class PhoneApp : public wxApp
{
public:
  bool OnInit() {
    if (!wxApp::OnInit()) return false;

    PhoneFrame* frame = new PhoneFrame();
    frame->Show();
    return true;
  }
};

IMPLEMENT_APP(PhoneApp)
```

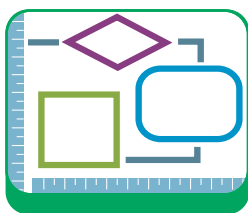
CONCLUSIONI

In questo primo appuntamento abbiamo visto come wxWidgets gestisce gli eventi e le applicazioni, e a questo punto vi rimane solo da sperimentare, con l'aiuto della guida in linea. Provate a creare finestre, pulsanti e controlli di ogni tipo e gestirne gli eventi. Appuntamento al prossimo mese: dobbiamo ancora fare tanti passi avanti sulla strada di wxWidgets!

Roberto Allegra

L'AUTORE

Il sito www.robertoallegra.it contiene l'elenco degli articoli pubblicati in questa rubrica, con gli inevitabili approfondimenti ed errata corrige. L'e-mail dell'autore è posta@robertoallegra.it.

FARE E DISFARE CON IL "COMMAND"

DOVETE IMPLEMENTARE UN SISTEMA DI "UNDO", ESEGUIRE ISTRUZIONI REMOTE O SEMPLICEMENTE ORGANIZZARE DEI MENU? IN TUTTI QUESTI CASI, IL PATTERN COMMAND PUÒ ESSERE LA SOLUZIONE CHE STATE CERCANDO

Sembrava uno dei soliti lavori: semplici correzioni ad un software già esistente. Invece Beatrice, la brava programmatrice, si è trovata davanti ad una massa di codice inestricabile come un piatto di spaghetti, scritto da un ex programmatore che nel frattempo si è licenziato per aprire un bar sulla Riviera.
L'applicazione è un semplice programma per gestire liste di nomi (in realtà il vero programma fa molte altre cose – ma volete un esempio semplice, vero?). Una funzionalità centrale del programma è l'"Undo": qualsiasi cosa si faccia, è sempre possibile tornare indietro. Purtroppo, il codice dell'Undo somiglia ad un allevamento di bacherozzi.

FARE E DISFARE

Per implementare Undo, l'autore del codice ha escogitato un sistema piuttosto complicato. Ciascun comando nel menu ha un numero identificativo. L'utente seleziona un comando inserendo il suo numero. Il programma esegue il comando e ne salva l'identificativo in un array. Quando l'utente chiede di annullare l'ultimo comando, il programma pesca l'ultimo identificativo nell'array e applica il comando inverso. Vediamo il codice:

```
$names = []
$commands = []
```

$names è un array globale (il programma usa il linguaggio Ruby, che identifica le variabili globali con il prefisso $). *$names* contiene la lista di nomi, che inizialmente è vuota. Una seconda variabile globale, *$commands*, contiene un array di numeri. Ciascun numero identifica un comando che l'utente ha selezionato dal menu. (Ovviamente Beatrice sa che le variabili globali sono una brutta cosa, e ha intenzione di toglierle di mezzo appena possibile – ma questo è un lavoro da rimandare ai prossimi giorni).
Il pezzo di programma che si occupa del menu definisce invece una variabile menu che contiene le varie voci da presentare all'utente:

```
menu = ["0 - Aggiungi nome", "1 - Inverti lista", "2 -
                                    Annulla", "3 – Esci"]
```

Quindi l'utente può dare il comando 0 per aggiungere un nome alla lista e 1 per invertire la lista. Se l'utente sceglie 2, l'ultima operazione viene annullata. Si può anche scegliere 2 più volte di seguito per annullare una serie di operazioni. Infine, scegliendo 3 si esce dal programma. Il cuore del programma è un ciclo *while* infinito che chiede all'utente di scegliere un comando e lo esegue:

```
while true
  puts $names.inspect
  puts menu.join(', ')
  command_number = gets.to_i
  case command_number
  when 0
    puts "Inserisci il nuovo nome:"
    name = gets.chomp
    $names << name
    $commands << command_number
  when 1
    $names.reverse!
    $commands << command_number
  when 2
    undo
  when 3
    puts "Ciao, ciao..."
    exit
```

COME LEGGERE QUESTO ARTICOLO

In questo articolo usiamo alcune funzionalità avanzate del linguaggio Ruby, che non tutti conoscono. Non preoccupatevi: non dovete capire ogni singola riga, e quando vedete che qualche dettaglio non viene spiegato, questo significa che non è importante. Immaginate di guardare il codice da sopra la spalla di Beatrice, che vi spiega a grandi linee come funziona. La cosa importante è che capiate il principio su cui si basa il pattern Command.

```
  end
end
```

Se non siete programmatori Ruby, non perdete tempo a capire tutti i dettagli di questo codice – l'importante è che capiate cosa fa. La prima istruzione del ciclo stampa l'attuale lista dei nomi con il metodo puts (il metodo *inspect* restituisce una versione dell'array buona per essere stampata sullo schermo, con parentesi quadre e virgole tra gli elementi). La seconda istruzione stampa il menu, con le varie voci separate da una virgola (questa volta usiamo il metodo *join*, perché non vogliamo le parentesi quadre). La riga successiva chiede all'utente (con il metodo *gets*) di inserire l'identificativo di un comando. Questo identificativo viene convertito in un numero intero con il metodo *to_i*.

Ora che conosciamo l'identificativo del comando, possiamo eseguirlo. Questo compito è svolto da un grosso *case*, l'equivalente Ruby dell'istruzione *switch* di Java e C. Vediamo meglio cosa succede se l'utente sceglie l'operazione *0*:

```
puts "Inserisci il nuovo nome:"
name = gets.chomp
$names << name
$commands << command_number
```

Il programma chiede all'utente di inserire il nuovo nome da aggiungere alla lista. L'input dell'utente viene passato attraverso il metodo *chomp*, che taglia via il carattere di a-capo finale, e poi aggiunto alla fine dell'array di nomi (questo è il significato di <<). Ma non basta: dobbiamo anche conservare il codice del comando nell'array dei comandi eseguiti. Qualcosa di simile succede anche se l'utente sceglie il comando *1*: la lista viene invertita, e il codice del comando finisce nell'array *$commands*. I comandi *3* e *4* sono speciali. Non possono essere annullati, quindi non vengono conservati nell'array dei comandi. Il comando *4* stampa un messaggio di saluto e termina il programma. Il comando *3* chiama il metodo *undo*:

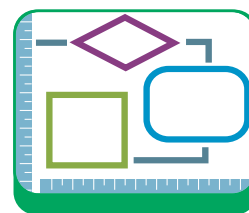
```
def undo
  return if $commands.empty?
  case $commands.pop
  when 0
    $names.pop
  when 1
    $names.reverse!
  end
end
```

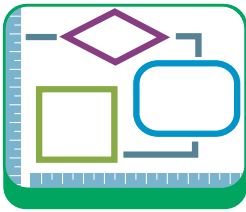
In Ruby si può mettere un *if* alla fine di una riga anziché all'inizio. Quindi la prima riga di questo metodo significa: "esci subito se l'array dei comandi eseguiti è vuoto". In caso contrario, l'identificativo dell'ultimo comando eseguito viene estratto dall'array dei comandi con il metodo *pop*. A questo punto arriva un altro *case*, che è l'esatto opposto del precedente: anziché eseguire ciascun comando, esegue il suo inverso. Annullare il comando *0* significa eliminare l'ultimo nome dalla lista, e annullare *1* significa invertire nuovamente la lista. Ecco cosa succede se facciamo girare il programma e gli diamo una serie di comandi (le righe in **grassetto** sono il nostro input, il resto è l'output del programma):

```
[]
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
0
Inserisci il nuovo nome:
Gino
["Gino"]
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
0
Inserisci il nuovo nome:
Piera
["Gino", "Piera"]
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
1
["Piera", "Gino"]
```

Abbiamo aggiunto due nomi e invertito la lista. Ora torniamo indietro:

```
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
2
["Gino", "Piera"]
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
2
["Gino"]
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
2
[]
0 - Aggiungi nome, 1 - Inverti lista dei nomi, 2 -
                                        Undo, 3 - Esci
3
Ciao, ciao...
```

Quindi l'idea alla base di questo codice funziona. Purtroppo, però, è facilissimo commettere errori quando si aggiungono nuovi comandi. E infatti Beatrice si trova davanti a problemi di ogni genere nel codice del programma (che ovviamente contiene più comandi di quelli che abbiamo mostrato): voci di menu che dimenticano di salvare il co-

UNA PICCOLA SFIDA

Se conoscete un po' di Ruby, potete provare a completare questo articolo aggiungendo una funzionalità di Redo che riapplica in sequenza le operazioni annullate. È un esercizio che vi richiederà qualche minuto, ma vi darà anche soddisfazione.

mando corrispondente nell'array; comandi che qualcuno si è dimenticato di aggiungere al *case* nel metodo *undo*, e quindi generano un errore quando si cerca di annullarli; errori nella corrispondenza tra quello che viene scritto nel menu e il comando che viene eseguito; e così via.
Beatrice non è tipa da farsi scoraggiare. Si rimbocca le maniche e si mette in cerca di una soluzione.

UN PO' DI OGGETTI

La prima cosa da fare, pensa Beatrice, è applicare un po' di programmazione object-oriented. Il sistema di partenza identifica ciascun comando con un numero. Se questi comandi fossero degli oggetti, non ci sarebbe bisogno di ricordare a quale numero corrisponde ciascun comando: basterebbe conservare un riferimento all'oggetto. Ad esempio, il comando che aggiunge un nome potrebbe diventare una classe fatta così:

```
class NewName
  def execute
    puts "Inserisci il nuovo nome:"
    $names << gets.chomp
  end

  def undo
    $names.pop
  end
end
```

Ora non c'è bisogno di identificativi. Si può semplicemente creare un oggetto di classe *NewName*, e questo oggetto ha un metodo che applica il comando e un secondo metodo che lo annulla. Si può fare lo stesso con il comando che inverte la lista, che è anche più semplice:

```
class ReverseNames
  def execute
    $names.reverse!
  end

  def undo
    $names.reverse!
  end
end
```

Si possono incapsulare in due classi anche i comandi "Undo" ed "Exit". Questi due comandi non possono essere annullati, quindi hanno un metodo *execute* ma non un metodo *undo*:

```
class Undo
  def execute
    $commands.pop.undo unless $commands.empty?
  end
end

class Exit
  def execute
    puts "Ciao, ciao..."
    exit
  end
end
```

L'istruzione unless è il contrario di *if* – significa "a meno che". Quindi il metodo *execute* della classe *Undo* significa: estrai l'ultimo elemento dall'array dei comandi e chiama il suo *undo*, a meno che l'array non sia vuoto.
Ora Beatrice può modificare il programma principale perché usi questi oggetti. Per cominciare, può affiancare all'array che contiene le voci di menu un secondo array che contiene le operazioni:

```
menu = ["0 - Aggiungi nome", "1 - Inverti lista", "2 -
                                   Annulla", "3 - Esci"]
commands = [NewName.new, ReverseNames.new,
                                   Undo.new, Exit.new]
```

NewName.new crea un oggetto di classe *NewName*, e così via. Quindi *commands* contiene un'istanza di ciascun comando. A questo punto, il programma principale diventa semplicissimo:

```
while true
  puts $names.inspect
  puts menu.join(', ')
  command = commands[gets.to_i]
  command.execute
  $commands << command if command.respond_to?
                                              :undo
```

IL COMMAND NEL MONDO REALE

Nel mondo Java, non dobbiamo cercare molto per trovare esempi di Command.

Prevayler **(http://www.prevayler.org) è un sistema di persistenza piuttosto originale. Anziché mettere gli oggetti in un database, Prevayler li serializza in un file (lo "snapshot"). Ciascuna modifica agli oggetti viene anch'essa immediatamente serializzata in un file prima di essere applicata. Quando si riavvia il sistema, Prevayler legge lo snapshot e poi riapplica tutte le modifiche. Ma ciascuna modifica deve a sua volta essere un oggetto, per poterla salvare su file e in seguito ricaricare ed eseguire. Questi oggetti, che Prevayler chiama "transazioni", sono dei Command.**

Struts **(http://struts.apache.org) è stato per anni il più popolare framework per sviluppare applicazioni Web. Al centro di Struts c'è il concetto di "action" - le azioni che vengono richieste dal client ed eseguite sul server. Per esempio, una action può validare un login o aggiungere un commento ad un blog. Anche le action, indovinate un po', non sono che implementazioni del buon vecchio pattern Command.**

```
end
```

Questo codice usa il numero scritto dall'utente come un indice per pescare il comando giusto dall'array dei comandi. Poi esegue il comando chiamando il suo metodo *execute*. Infine aggiunge ilo comando all'array dei comandi – ma solo se il comando può essere annullato. Questa riga di codice, un po' esoterica per chi non programma in Ruby, significa: "aggiungi questo comando all'array dei comandi, ma solo se il comando ha un metodo che si chiama *undo*". Quindi gli oggetti di classe *NewName* e *ReverseNames* finiscono nella lista dei comandi, *Undo* ed *Exit* no.
Beatrice fa girare questa nuova versione del programma, e ottiene esattamente lo stesso risultato della versione originale. Successo! La nostra amica, però, non è tipa da lasciare un lavoro a metà.

SEMPRE PIÙ OBJECT-ORIENTED

Per aggiungere una nuova operazione nel menu, riflette Beatrice, si devono ancora modificare due array: *menu* e *commands*. È ancora facile sbagliarsi, ad esempio aggiungendo una voce al *menu* senza il corrispondente *command*. Ora che ciascun comando è un oggetto, pensa Beatrice, non sarebbe più comodo inserirlo direttamente nel menu?
L'unico problema di questo approccio è che i comandi devono contenere anche la stringa che appare sul menu. Ma questa è una modifica facile: basta aggiungere a ciascun comando un metodo che lo trasforma in una stringa quando lo si stampa. In Ruby, questo avviene automaticamente se si scrive un metodo *to_s* (l'equivalente Java è il metodo *toString()*):

```
class NewName
  ...

  def to_s
    "Aggiungi nome"
  end
end

class ReverseNames
  ...

  def to_s
    "Inverti lista"
  end
end

class Undo
```

```
  ...

  def to_s
    "Annulla"
  end
end

class Exit
  ...

  def to_s
    "Esci"
  end
end
```

Ora i comandi sono anche voci di menu, e gli array *menu* e *commands* possono essere fusi in un solo array:

```
menu = [NewName.new, ReverseNames.new,
                          Undo.new, Exit.new]
```

Dato che la stampa del menu sullo schermo deve anche associare un numero a ciascun comando, Beatrice scrive un metodo che stampa il menu facendo precedere ciascun comando dalla sua posizione nell'array (questo metodo vi sembrerà delirante se non siete programmatori Ruby, ma fidatevi – funziona):

```
def show(menu)
  idx = -1; puts menu.map {|command| idx += 1;
               "#{idx} - #{command}"}.join(', ')
end
```

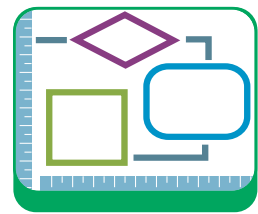
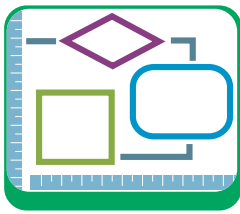
SE AL TUO LINGUAGGIO PIACCIONO I TIPI

Cosa succede se anziché usare un linguaggio "dinamico" come Ruby volete implementare il pattern Command in un linguaggio "statico" come Java? Rivediamo uno dei Command dell'articolo:

```
class Exit
  def execute
    puts "Ciao, ciao..."
    exit
  end
end
```

Nel nostro esempio, la classe Exit non è obbligata ad ereditare da nessuno. Possiamo semplicemente creare un Exit, passarlo in giro per il codice, e chiamare execute in qualsiasi punto del programma. In Java, siamo obbligati a dichiarare i tipi: il codice che chiama execute vuole sapere con certezza che questo metodo esiste. Questo significa che dobbiamo avere una classe o un'interfaccia comune a tutti i comandi:

```
public interface Command {
  public void execute();
}
public class Exit implements Command
                                   {
  public void execute(){
    System.out.println("Ciao, Ciao...");
    System.exit(0);
  }
}
```

Ora il codice che deve eseguire o annullare qualsiasi comando può parlare all'interfaccia Command. Come sempre, avere un linguaggio dove si dichiarano i tipi significa scrivere un po' di codice in più, per avere in cambio più controllo da parte del compilatore.

Infine, ecco la nuova versione del programma principale. È simile alla precedente, ma usa il nuovo array unico per i menu:

```
while true
  puts $names.inspect
  show(menu)
  command = menu[gets.to_i]
  command.execute
  $commands << command if command.respond_to?
                                          :undo
end
```

Ora il lavoro è davvero finito. Per aggiungere un nuovo comando basta creare una classe che lo incapsula, definirne i metodi *execute, to_s* ed eventualmente undo e infilare un oggetto di questa classe nell'array *menu*. È quasi impossibile sbagliare. Beatrice può tornare a casa soddisfatta.

IL PATTERN COMMAND

L'idea di Beatrice è semplice e potente: se ho un sistema che manipola comandi, posso incapsulare questi comandi in una serie di oggetti, ciascuno con un metodo che esegue il comando. Questa è l'essenza del design pattern *Command*.
Quando hai degli oggetti al posto delle normali righe di codice, ti si apre tutto un mondo di possibilità. Ne abbiamo appena viste alcune: i Command possono essere usati direttamente in un menu, o messi da parte per poterli poi annullare. Ma ci sono molti altri modi creativi per usare il pattern Command. Ad esempio potrei scrivere un registratore di macro che conserva i comandi per poi eseguirli di nuovo. Oppure posso costruire uno scheduler che accoda i comandi per eseguirli in seguito, magari in orari predefiniti o con diverse priorità. Oppure posso semplicemente convertire tutti i comandi in stringa e stamparli in un log.. Un campo dove questo pattern è particolarmente utile sono le comunicazioni di rete. Un client può creare un Command, serializzarlo e spedirlo al server. Il server riceve il comando e lo esegue. In questo modo potete scrivere un sistema distribuito che può essere facilmente esteso con nuovi comandi (ovviamente il client deve essere affidabile, o questo potrebbe essere un pericoloso "buco" di sicurezza).Insomma: tutte le volte che volete manipolare pezzi di codice, passarli in giro, salvarli da qualche parte, e così via, ricordate del pattern Command.

UN COMANDO SENZA CLASSE

Quando si deve scrivere un Command, si scrive di solito una classe con metodo execute. In realtà la parte interessante del Command è il metodo *execute*, non la classe. La classe esiste solo per avvolgere l'operazione in un oggetto.
In molti linguaggi, come in Java, questo "impacchettamento" è indispensabile: non si può avere un'operazione senza un oggetto che la ospiti. In altri linguaggi, come in Ruby, esistono alternative. In Ruby, si può passare direttamente un pezzo di codice ad un metodo. Questa è una funzionalità avanzata, ma vale la pena di spenderci qualche parola:

```
$commands = []

def record(&command)
  $commands << command
end
```

Il metodo *record* prende un parametro speciale, identificato dal prefisso &. Questo parametro è un pezzo di codice tra parentesi graffe – quello che Ruby chiama un *blocco*. Il blocco viene automaticamente convertito da Ruby in un oggetto di classe *Proc* (che sta per "procedura"), che il metodo *record* aggiunge poi ad un array globale di comandi. Quindi possiamo usare direttamente il blocco come un Command, senza definire nessuna classe:

```
record { puts "Buongiorno!" }
record { puts Time.now }
record { puts "Arrivederci!" }
```

Le tre stampe che abbiamo inviato a *record()* non vengono eseguite subito, ma convertite in *Proc* e conservate nell'array. Possiamo eseguire le *Proc* più avanti, chiamando il loro metodo *call()*:

```
for c in $commands
  c.call
end
```

Il risultato è:

```
Buongiorno!
Tue Jul 10 11:39:12 +0200 2007
Arrivederci!
```

CONCLUSIONI

Il Command è uno dei pattern più flessibili e comuni. È anche molto semplice, ma non fatevi ingannare: dietro questa semplicità si nascondono molte sfumature, come ha imparato Beatrice.
Arrivederci al mese venturo, con un altro problema e un'altra soluzione!

Paolo Perrotta

SOFTWARE SUL CD

Java SE Development Kit 6U2

IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

ARGOUML 0.24

IL DISEGNATORE DI DIAGRAMMI

Di Argo UML ce ne parla in modo approfondito in questo stesso numero di ioProgrammo, Massimiliano Bigatti, all'interno del corso UML. Si tratta di un ottimo editor che aiuta nella realizzazione di progetti UML. Consente facilmente di identificare oggetti, classi, azioni all'interno del progetto e poi relazionarli fra loro come si compete ad ogni editore UML che si rispetti. Si tratta di un ottimo prodotto multipiattaforma.

Directory:/argouml

CRIMSON EDITOR 3.7.0

UN EDITOR LEGGERE E COMPLETO

Crimson Editor è un prodotto piuttosto interessante per l'editing del codice in ambiente Windows. Unisce due caratteristiche di tutto rispetto, ovvero quella di essere ultraleggero e di disporre di Syntax HighLighting per quasi tutti i software oggi in circolazione. HTML, C/C++, Perl, Java, Matlab and LaTeX Può essere esteso anche per altri linguaggi utilizzando un file di configurazione apposito. Ovviamente contiene anche tutte le altre caratteristiche di un normale editor per programmatori, undo/redo, macro etc e infine come ultima nota c'è da aggiungere che occupa uno spazio veramente ridotto tanto che può essere trasportato su una penna USB che abbia veramente poco spazio a disposizione

Directory:/crimsoneditor

DJANGO 0.9.6

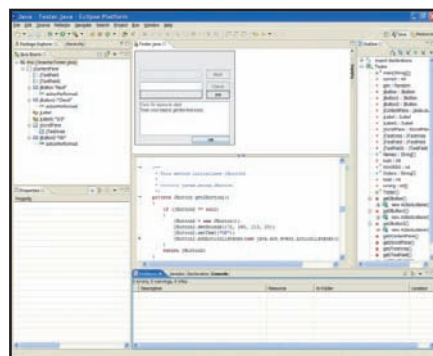
IL FRAMEWORK EMERGENTE

Django è un progetto che sta facendo parlare di se negli ultimi tempi. Si tratta di un framework basato su Python che unisce flessibilità a potenza. Il concetto è quello tipico dei compilatori per il web. Si crea una certa struttura su disco, si editano alcuni file con una sintassi particolare, si da tutto in pasto al compilatore che tira fuori una splendida applicazione Web, per l'occasione basata su python. La cosa interessante sta nella facilità della curva d'apprendimento e nella completezza del linguaggio

Directory:/django

ECLIPSE PER JAVA E C++

L'IDE TUTTOFARE

Eclipse è ormai un must in programmazione. Si tratta dell'IDE buono per ogni occasione. Una piattaforma estendibile all'interno della quale trovano spazio specializzazioni rivolte a Java come a C++ come a PHP In pratica una sorta di scheletro di IDE dedicato ai programmatori sopra il quale si possono costruire applicazioni specializzate per ogni tipo di linguaggio.

In questo numero vi presentiamo l'IDE dedicato ai programmatori Java e la novità appena rilasciata dedicata ai programmatori C++

Directory:/eclipse

EDIT PLUS 2.31

L'EDITOR PRONTO PER INTERNET

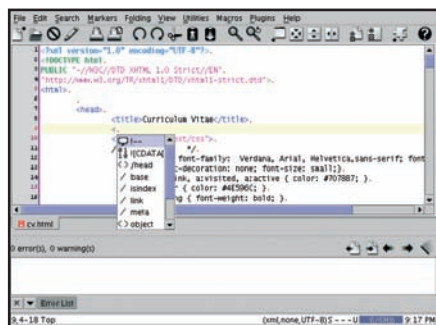
Internet-Ready, questo è il motto che contraddistingue Edit Plus. Si tratta di

un ottimo IDE che sostituisce tranquillamente il notepad ma vi aggiunge la Syntax Highlighting HTML, CSS, PHP, ASP, Perl, C/C++, Java, JavaScript Inoltre contiene un FTP client per l'upload delle pagine, un potente motore di ricerca e molto altro.

Directory:/editplus

JEDIT 4.3

IL LEGGERO MULTIPIATTAFORMA

Jedit è un editor per programmatori giunto ormai alla sua piena maturità. Dispone di centinaia di caratteristiche utili ai programmatori e può essere estesto tramite plugin. Inoltre si tratta di un editor completamente multipiattaforma. Tra le varie caratteristiche si annotano la presenza di un linguaggio interno per lo sviluppo di Macro, la possibilità di essere esteso tramite plugin, l'autoIndent e la Syntax Highlighting. Infine bisogna dire che supporta circa 130 linguaggi.

Directory:/jedit

ANTECHINUS JAVASCRIPT EDITOR 9.0

L'EDITOR PER L'AUTOMAZIONE DEL WEB

Sia che siate programmatori professionisti, sia che siate noviz, questo editor JavaScript può migliorare di molto la vostra esperienza come programmatori di siti web.
Si tratta di un editor specifico per JS che implementa alcune automazioni. Ad esempio è possibile aggiungere facilmente un menu rollover alla pagine, oppure il codice di validazione delle form, slide, calendari, selettori di colore e così via. Si tratta realmente di un editor interessante, da provare per chi programma quotidianamente applicazioni per il Web.

Directory:/jseditpro

LIGHTSTREAMER

AJAX IN TECNOLOGIA PUSH

Di questo prodotto ce ne ha parlato in modo veramente molto approfondito Enrico Viale nello scorso numero di ioProgrammo. Si tratta di un framework che inverte il normale flusso di lavoro di una catena Ajax. Mentre di solito è il client a fare una richiesta e ad aggiornare il pezzo di pagina interessato, nel caso di LightStreamer la pagina si aggiorna anche senza una richiesta specifica del client ma con un invio in push di una richiesta di aggiornamento da parte del server sulla base di un evento. Ad esempio quando si inserisce un nuovo record in un database

Directory:/lightstreamer

LSL EDITOR

L'IDE PER SECOND LIFE

Di Second Life si sente parlare ormai in ogni dove. L'universo parallelo ideato da Linden Lab contiene sempre più di tutto. Appartamenti, negozi, spazi virtuali, oggetti di ogni tipo. La cosa interessante è che tutta questa roba è programmabile ed estendibile. Siete voi programmatori che utilizzando il Linden Language date vita a questo mondo. LSL-Editor è un editor specializzato per il Linden Language che vi consentirà di animare facilmente le vostre creazioni nell'universo virtuale

Directory:/lseditor

MYSQL 6.0 BETA

IL DB PER LA RETE

Indispensabile per programmare web application in tecnologia PHP. Non che non sia possibile utilizzare altri database, ma MySQL e PHP rappresentano veramente un binomio inscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti

Directory:/mysql60

NOTEPAD++ 4.1.2

A prima vista Notepad++ sembrerebbe un normale editor di testo, uno dei tanti in circolazione; ma la caratteristica che lo differenzia dagli altri software del suo genere è la struttura, in ambiente Windows che supporta numerosi linguaggi di programmazione e, soprattutto, è scritto interamente in C++. Le sue funzioni sono numerose, atte a soddisfare le esigenze di qualunque tipo di programmatore; sono presenti le tabulazioni, la compressione, l'espansione della struttura, la conversione in formato UNIX, Windows e MAC, la codifica in ANSI, UTF-8 e UCS-2, le macro e i plugins. Queste sono solo alcune funzioni; infatti all'interno dell'applicazione troviamo l'opzione di Find, la stampa, la possibilità di aprire più sessioni, la scelta del linguaggio (ben 38 diverse forme) e la configurazione dello stile e dei tasti di scelta rapida.

Directory:/notepadpp

OPEN COMPUTER VISION LIBRARY 1.0

Più che una semplice libreria per la gestione degli oggetti grafici. Si tratta di un'insieme di API piuttosto futuristiche, si va da riconoscimento del movimento a quello facciale, all'isolamento dei contorni. Si tratta di una libreria immensa che offre un quantitativo sterminato di funzioni relative al multimedia e alla grafica

Directory:/opencv

PHP 5.2.3

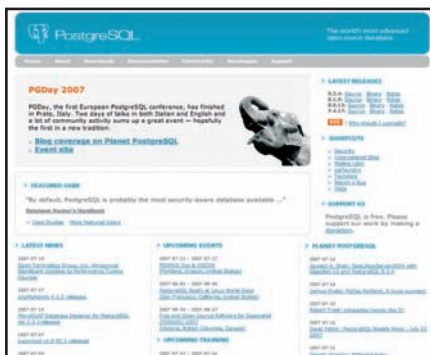
Sono tre le colonne portanti di Internet: PHP, APACHE e MySQL. Certo la concorrenza è forte. Asp.NET e SQL Server avanzano con celerità, ma a tutt'oggi non si può affermare che i siti sviluppati in PHP costituiscano la stragrande maggioranza di Internet. Quali sono le ragioni del successo di cotanto linguaggio? Prima di tutto la completezza. PHP ha di base tutto quello che serve ad un buon programmatore, raramente è necessario ricorrere a librerie esterne, e quando è proprio indispensabile farlo esistono comunque una serie di repository che rendono tutto immediatamente disponibile ed in forma gratuita. Il secondo punto di forza del linguaggio sta nella sua capacità di poter essere utilizzato sia in modo procedurale che nella sua

forma ad oggetti certamente più potente e completa. Esiste un terzo di punta di forza essenziale che è quello riguardante la curva di apprendimento. PHP è in assoluto uno dei linguaggi con la curva di apprendimento più bassa nel panorama degli strumenti di programmazione. Si tratta perciò di uno strumento indispensabile per chi si avvvicina alla programmazione web, a meno che non intendiate scegliere strade diverse quali possono essere ASP.NET o JSP

Directory:/php523

POSTGRESQL 8.2.4

IL GRATIS COMPLETO E VELOCE

Nessun server di database offre la completezza delle funzioni esposte da PostgreSQL e rimane comunque completamente Gratis. PostgreSQL è probabilmente il più estendibile fra i database esistenti. Inutile parlare della gamma praticamente completa delle sue funzioni. Il punto di forza che lo pone probabilmente al di sopra di tutti i concorrenti rimane l'alta possibilità di personalizzazione, oltre, naturalmente, alla velocità, alla stabilità ed al costo nullo. Unica pecca, una certa complessità nella gestione. E' sicuramente da usare in ambienti di produzione professionali che non possono accontentarsi di alcun compromesso

Directory:/postgres

PYTHON 2.5

L'EX GIOVANE RAMPANTE

Python è stato considerato per lungo tempo il nuovo che avanza. Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione in un gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet. Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows

Directory:/python

SPICES. DECOMPILER 5.1.4.4

IL DECOMPILATORE PER .NET

Interessante questo decompilatore /disassemblatore che sfrutta l'MS Intermediate Language per risalire da un binario al corrispondente sorgente. Tra le altre cose il software ricostruisce anche il diagramma di flusso dell'applicazione. FUnziona con C#,VB.NET, Delphi.NET, J# e managed C++

Directory:/spices

WXWIDGETS 2.8.4

COMODE LIBRERIE PER LO SVILUPPO DI INTERFACCE GRAFICHE

Interessantissime queste librerie, più volte le abbiamo utilizzate all'interno di ioProgrammo per realizzare degli esempi. Si tratta di librerie che consentono la creazione di interfacce grafiche, possono essere utilizzate da C++ ma anche da altri linguaggi come ad esempio Python. La cosa estremamente interessante è che consentono lo sviluppo di applicazioni completamente multipiattaforma, sono disponibili infatti sia in ambiente unix che in ambiente Windows

Directory:/wxwidhets

AJAX WEBSHOP 3

IL RAD PER AJAX

JoyiStar AJAX WebShop è un interessantissimo editor Visuale per Ahax. Funziona a componenti come la maggio parte dei RAD e consente di sviluppare una completa applicazione Ajax semplicemente trascinando sul Desktop virtuale l'elemento che meglio si adatta alle esigenze del programmatore. Si tratta realmente di un'ottima idea, considerata la dimensione che Ajax sta sempre maggiormente assumendo nello sviluppo Internet

Directory:/webshop

SYNEDIT 2.0.5

L'EDITOR PER BORLAND

Synedit è un editor che ben si concilia con i prodotti Borland, in particolare C++ Builder, Delphi e Kylix. Supporta la sintax highlighting e wil word-wrap. Include caratteristiche interessanti come i template e la code completion, intolre consente di esportare in formato tex, html ed RTF

Directory:/synedit

TEXTPAD 4.7.3

IL PLURIPREMIATO EDITOR PER WINDOWS

Leggero, colmo di funzionalità, con supporto avanzato a qualunque tipo di linguaggio. Sono queste le caratteristiche che nel corso del tempo hanno fatto di TextPad uno degli editor più amati e conosciuti dai programmatori. Utilissimo quando non si vuole ricorrere ad un completo IDE ma si vuole disporre di funzioni avanzate di ricerca, syntax highlighting e code complexion

Directory:/textpad

ULTRAEDIT-32

OTTIMO EDITOR TESTUALE

UltraEdit-32 è un ottimo editor testuale. L'applicazione supporta numerosi linguaggi di programmazione, come PERL, HTML,JAVA,C/C++, FORTRAN e molti altri ancora. La sua interfaccia grafica è suddivisa in due finestre: una dedicata alla ricerca e all'apertura dei file; l'altra rivolta alla visualizzazione del documento selezionato. Le sue caratteristiche sono numerose, così come i funzionamenti: invio file tramite FTP, programmazione macro, funzione di ricerca, costruzione stile, modifica riga e colonna, supporto per SFTP, SSH/TELNET, UNICODE, conversione UNIX/MAX, formattazione testi, rinomina, anteprima di stampa, inserimento e duplicazione linee, realizzazione di Bookmarks.

Directory:/ultraedit

CRIVELLO QUADRATICO

IL CRIVELLO QUADRATICO È UNO DEI PIÙ IMPORTANTI ED EFFICIENTI METODI PER LA FATTORIZZAZIONE DI NUMERI. MA SOPRATTUTTO UNO STRUMENTO UTILIZZATO PER LA CIFRATURA CON RSA. ANALIZZIAMONE IL FUNZIONAMENTO

Nel campo della sicurezza informatica si susseguono gli sforzi per individuare metodi e strumenti sempre più avanzati che proteggano ancora di più e meglio i nostri dati. Come è capitato di constatare in diverse occasioni, le energie profuse in questo ambito sono ingenti, perché è particolarmente sensibile il problema della trasmissione e del mantenimento dei dati in piena affidabilità. I metodi a chiave pubblica come il noto RSA, oggi molto diffusi, fondano la propria azione su semplici quanto potenti proprietà dei numeri interi. Ecco che il lavoro del matematico risulta particolarmente prezioso per sviluppare detti procedimenti. Demolire un sistema del tipo RSA, in estrema sintesi, significa trovare due numeri che moltiplicati fra loro producono un altro numero conosciuto ma estremamente grande. È questo un problema di fattorizzazione. Ad esempio, RSA 2048 è un numero di 617 cifre decimali. Numeri talmente grandi e proprietà algebriche di base ad essi associati, come la fattorizzazione o la ricerca del massimo comun divisore, sono prevalentemente usati nel campo della sicurezza. Se si pensa che la fisica, che storicamente trattava numeri considerati "ostici" da manipolare, estremamente piccoli o estremamente grandi, non va oltre valori dell'ordine di 10 alla 92, ossia numeri con novantadue cifre, ci rendiamo conto che di fatto il primato sul trattamento di numeri di grandi dimensioni è da assegnare all'ambito della sicurezza informatica. Avere a che fare con numeri di grandissime dimensioni è un modo per impegnare per tempi significativi i computer nonostante le eccezionali performance che peraltro sappiamo essere in continua crescita. Ad esempio, in una sfida, di qualche anno, si chiedeva di fattorizzare un numero RSA-576, un numero decimale con 174 cifre, il vincitore avrebbe intascato un premio di 10 mila dollari offerti dall'agenzia RSA security. La sfida è stata vinta nel 2004 con un lavoro di equipe tra l'istituto nazionale di matematica pura tedesco e il centro nazionale di ricerca e scienza del computer olandese che hanno impegnato per più di tre mesi 100 computer. Quindi nonostante sia stata vinta la sfida, per i tempi e gli strumenti impegnati gli utenti RSA si sentono ulteriormente garantiti. In questo appuntamento ci dedicheremo al problema matematico della fattorizzazione sapendo che alla base di molti sistemi di crittografia. In particolare analizzeremo il crivello quadratico, tra i più efficienti oggi disponibili, un metodo chiave introdotto da Pomerance che aiuta a comprendere l'intero filone che afferisce alla fattorizzazione. Introdurremo nei paragrafi iniziali il percorso storico che ha condotto a tale metodo, visionando e approfondendo, anche con la produzione di codice, i primi procedimenti per fattorizzare.

IL PRIMO METODO DI FATTORIZZAZIONE

Come è stato detto, fattorizzare un numero vuol dire trovare un insieme di numeri primi che moltiplicati tra loro diano il numero stesso. Se n è il numero, allora tale fattorizzazione si può esprimere nella forma:

$$n = p_1^{np1} \cdot p_2^{np2} \cdot \ldots \cdot p_m^{npm}$$

I numeri p1, p2, .. ,pm sono i primi che compongono il numero, ognuno di essi si può rispettivamente presentare np1, np2, .. ,npm volte. Questo è il risultato a cui ambisce il matematico. Per l'informatico può essere sufficiente trovare due numeri p1 e p2 tali che n=x*y. Eventualmente si può procedere con la fattorizzione separata dei due numeri x e y. Ovviamente, bisogna subito controllare che non ci si trovi in casi limite come quello per cui n è un numero primo, quindi non scomponibile ulteriormente in fattori. Per controllare ciò si può usa-

re uno dei tanti test di primalità disponibili (si veda soluzioni di Fabio Grimaldi – ioProgammo n. 97). Un'altra situazione limite è che n sia il quadrato perfetto di un numero primo, anche in questo caso da un punto di vista informatico il numero è poco significativo. Ma veniamo al primo metodo introdotto per risolvere il problema. Esso è basato su tentativi che si attuano con ripetute divisioni. Il nome è divisione per tentativi. Si fa riferimento ad una serie di numeri primi fino ad un fissato valore, non molto grande, in letteratura si fissa questo valore a un milione. A riprova degli ordini di grandezza che stiamo trattando è utile soffermarsi sulla considerazione che un milione è considerato un numero piccolo, anzi molto piccolo. Quindi si comincia con i tentativi, si prova a dividere il numero per la lista di primi a disposizione. Ogni primo può dividere più volte il numero, si ottiene quindi una forma del tipo proposto prima:

$$n = p_1^{np1} \cdot p_2^{np2} \cdot \ldots \cdot p_m^{npm} \cdot k$$

Con k fattore non ancora scomposto. Ovviamente si auspica che k non sia presente, qualora ci sia è necessario riferirsi a qualche altro algoritmo per fattorizzare k. Per avere la certezza di scomporre completamente n bisognerebbe costruire la successione di primi in un insieme di grandezza, in tal caso il numero di divisioni aumenterebbe in modo esponenziale sancendo la non efficienza del metodo.

IL CONTRIBUTO DI FERMAT

Fermat che conosciamo bene per la sua estrema dimestichezza con i numeri mise appunto un metodo per risolvere la questione proposta. La genesi di tale procedimento è ancora una volta il frutto di una sfida, questa volta con un altro solito noto: Mersenne. In una missiva questi provocava Fermat chiedendogli di fattorizzare il numero 100.895.598.169. Per l'epoca tali circostanze erano comuni, ed è grazie ad esse che la matematica ha visto una sostanziale evoluzione. L'idea di Fermat è stata di scrivere il numero n come la differenza di quadrati.

```
n=x2-y2 = (x+y)(x-y)
```

l'obiettivo è raggiunto poiché tale forma garantisce anche la scrittura del numero come prodotto di due numeri, i binomi (x+y) e (x-y). Si tratta di individuare i due numeri. Il procedimento proposto è per tentativi. Si prova con un primo valore di x, vedremo poi come sceglierlo, e si verifica se lo scarto del numero n rispetto al quadrato di x è a sua volta un quadrato. Si prova, in altri termini la relazione scritta sopra. Se così non è si seleziona un nuovo valore di x e si continua. Il processo si ferma quando lo scarto genera un quadrato, abbiamo cioé trovato la y, quindi la soluzione. Entriamo adesso nel merito dell'algoritmo. Si scrive $n=x^2$-z. Bisognerà valutare diversi valori di x fin quando z non diventa un quadrato. Si comincia con ponendo x ad un valore iniziale vicino alla radice di n:

$$x = [\sqrt{n}] + 1$$

La parentesi quadra indica la parte intera. Ricordiamo che tratteremo solo con numeri interi.

Lo scarto o resto rispetto al quadrato è r. Facciamo i calcoli per verificare il più piccolo valore di x e z che può essere testato.

```
z=x2-n = (|n|+1)2-|n|2-r = 2|n|+1-r
x=|n|+1
```

Il passo successivo è testare la soluzione z. Siamo al termine dell'algoritmo qualora si tratti di un quadrato, altrimenti si provano nuovi valori di x (Fermat semplicemente proponeva di incrementare di uno) e verificare i conseguenti valori di z. Facciamo un esempio, lo stesso proposto da Fermat per spiegare il metodo. Sia n=2027651281 al primo passo si ottiene /n/=45029, r=40440, z=49619 e x=45030. Si verifica che z non è un quadrato. Si prova una seconda soluzione. Si incrementa la x e si ottiene conseguentemente un nuovo valore numerico di z.

```
z=(x+1)2-z=(x+1)2-x2+z=2x+1+z
z=2*45030+1+49619=139680
x=45031
```

Il numero z ancora non è quadrato, basta osservare che le ultime due cifre (80), esse non sono un quadrato. Si ricorda che se le ultime due cifre sono un quadrato si tratta di un quadrato, altrimenti no. Procedendo così ad un certo punto si ottiene un valore di z che è finalmente un quadrato, si tratta di z=1040400 che è quadrato di 1020. Riprendendo la relazione: n=x2-z, possiamo finalmente esprimere n come la differenza tra due quadrati le due radici corrispondenti sono *x=45041* e radice di z che è appunto 1020, che chiameremo *y*. Così i due fattori prodotto sono *(x+y)* e *(x-y)* ovvero *45041+1020=46061* e *45041-1020=44021*. In definitiva si può verificare che il risultato è:

```
n=(x+y)(x-y)=46061*44021= 2027651281
```

$$|n| = \lfloor \sqrt{n} \rfloor$$
$$n = |n|^2 + r$$

Una domanda di estrema attualità è: “chissà cosa mai avrebbe potuto produrre Fermat se avesse avuto a disposizione un calcolatore”. La codifica del suo metodo risulta alquanto agevole. Si tratta di predisporre un ciclo con il quale valutare sempre nuovi valori di x e riportare i risultati di interesse in output. Ecco la routine che attua il metodo. Si esce dal ciclo quando la soluzione è stata trovata. Per sapere fin quando cercare, quindi essere sicuri che si tratta di un numero primo è fondamentale, il contributo Lucas che partendo dal fatto che un numero primo deve necessariamente rispettare espressioni del tipo se $n=ab$ con $a<b$, ha trovato la relazione:

```
x+y=a, x-y=b e x=(a+b)/2=(a+n/a)/2
```

Questo ultimo numero è sicuramente più piccolo di *(n+1)/2* che a sua volta diventa un ottimo lower bound. Ossia, un limite oltre il quale è inutile controllare.
Solo nel caso degenere per cui n è primo si ha:

```
x+y=n, x-y=1
x=(n+1)/2
```

Ecco le due funzioni che consentono di implementare il metodo di Fermat.

```
...
const int nmax=15;
    long n,na,z,r,x;
     int nprove;
bool quadrato(long v)
{   long rv;
   rv=sqrt(v);
   return (rv*rv)==v;
};
void Fermat_e_Lucas(long nn)
{   long a,b,y;
   cout<<"\n\t"<<" z "<<'\t'<<" x "<<'\t'<<'\n';
   na=sqrt(nn);
   r=nn-na*na;
   x=sqrt(nn)+1;
   z=2*na+1-r;
   for (nprove=0;(!quadrato(z)) && (x<(n+1)/2);
                                          nprove++)
   {   z=z+2*x+1;
       x=x+1;
        cout<<'\t'<<z<<'\t'<<x<<'\n';
    };
    y=sqrt(z);
    a=x+y;
    b=x-y;
    cout<<"\n\t a:"<<a<<"\t b:"<<b;
    getch();
}; ... // di seguito realizzare il main program
```

quadrato restituisce vero se il numero è un quadrato. Per verificarlo si estrae prima la radice intera e poi si verifica se il quadrato è uguale al numero iniziale. La routine *Fermat_e_Lucas(long)* applica il metodo descritto. Per questi compiti il linguaggio C++ risulta molto efficiente. In assegnazioni del tipo rv=sqrt(v) si azionano le regole di casting. Attraverso conversioni implicite il valore a destra (r-value) viene automaticamente convertito nel tipo della variabile a sinistra (l-value), ossia un intero (long). In figura 1 è proposta l’esecuzione del programma per un caso specifico.

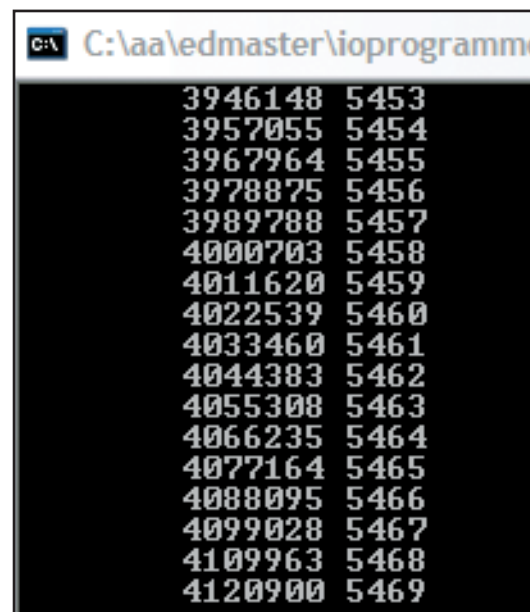



***Fig. 1:** “Output del programma sviluppato per l’implementazione metodo di fattorizzazione Fermat per n=25789061. Le due colonne di numeri indicano z e x”*

Il metodo visto può essere ottimizzato facendo opportune sostituzioni che ci evitano le costose computazioni di prodotti nel ciclo. Tuttavia si ha un pesante svantaggio che è il numero elevato di cicli.

METODI MISTI

Lo svantaggio del metodo di Fermat fu relativamente contenuto con l’algoritmo di Lehman che in definitiva è una via di mezzo dei due precedentemente visti. Si parte ponendo R pari alla radice cubica di n:

$R = \sqrt[3]{n}$

Si applicano così le divisioni per tentativi di tutti i primi minori o uguali di R. Potrebbe accadere che dopo i primi tentativi non si ottenga alcun risultato; in tal caso o n è primo oppure è esattamente il prodotto tra due numeri a e b primi, che sono contenuti nell'intervallo [R, R2] . A questo punto bisogna pensare un modo per trovare le soluzioni. Pomerance e Crandall hanno introdotto terne di numeri x, y e z interi che qualora n non sia primo rispettano la proprietà

$$x^2 - y^2 = 4kn \quad con \quad 1 \leq k \leq R$$

Per trovare *x* e *y* sono stati ideati opportuni intervalli che variano con *k*. Si inizializza con k=1 a cui corrispondono due valori di x, ossia x_0 e x_1

$$x_0 = [\sqrt{4kn}]$$
$$x1 = [\sqrt{4kn} + \frac{\sqrt{n/k}}{4R}]$$

Si cerca nell'intervallo $[x^0, x^1]$, se si verifica la prima condizione scritta ossia che x^2-4kn è un quadrato perfetto di y^2 allora la soluzione è il *MCD(x+y,n)*, altrimenti si incrementa k e si calcolano nuovi valori dell'intervallo dove ricercare le soluzioni. È stato dimostrato che la complessità di tale algoritmo è esponenziale. Miglioramenti di questo metodo sono stati introdotti dalla coppia di studiosi Lehmer e Powers che hanno sviluppato un procedimento con frazioni continue per ricercare più rapidamente le soluzioni nell'intervallo di interesse.
Il primo algoritmo efficiente è opera di D. Shanks che messe a punto SFF (Square form factorization). La soluzione che viene oggi usata quando si decide di usare i metodi con frazioni continue è opera di J. Brillhart e M. Morrison il cui algoritmo è conosciuto con la sigla CFRAC.

IL CRIVELLO DI POMERANCE

L'algoritmo proposto nel 1981 da Pomerance si basa sul concetto di congruenza quadratica e si sviluppa in due fasi. In una prima fase c'è la raccolta dei dati (data collection) e nella seconda fase l'elaborazione dei dati (data processing). La prima delle due fasi può essere opportunamente distribuita su un calcolo parallelo che ne velocizza sensibilmente la realizzazione, mentre la seconda deve essere elaborata in un'unica CPU, per questo si preferisce usare in questo stadio supecomputer.

Inizialmente l'approccio era totalmente improntato alla ricerca di una congruenza quadratica una relazione del tipo

$$x^2 \equiv y^2 (\bmod n) \qquad (1)$$

Ad esempio 802 mod 5959 è 441 ossia 21^2. Dall'esempio riportato si comprendono anche le notazioni con cui si riportano le congruenze. Per noi informatici che conosciamo *mod*, funzione dei linguaggi di programmazione, anche se ha lo stesso significato viene usata da un punto di vista tecnico diversamente. Una congruenza come quella esaminata raramente si ritrova per grandi valori di *n*. Certo qualora ci si potesse arrivare saremmo certi di aver terminato le nostre ricerche.
Per comprendere facciamo qualche esempio 41^2 mod 1649 = 32, 42^2 mod 1649 = 115, e 43^2 mod 1649 = 200. Notiamo con rammarico che nessuno dei tre esempi ha prodotto un quadrato, infatti dei risultati: 32, 115 e 200 nessuno è un quadrato perfetto. Si può però notare che $32*200=6400=80^2$,. Quindi mod 1649, $32*200=412*43^2$ che si può scrivere anche come $(41*43)^2$. Applicando il modulo si ottiene: 41*43 mod 1649 =114; questa è una congruenza quadratica che si scrive più precisamente come:

$$114^2 \equiv 80^2 (\bmod 1649)$$

Il problema cruciale è un altro: dato un insieme di numeri trovare un sotto insieme il cui prodotto è un quadrato. Osserviamo il percorso che ha condotto l'ideatore Pomerance allo sviluppo del metodo. La soluzione che si otterrà è un vettore esponente. Per esempio per il numero 5544 =23*32*7*11 il vettore esponente assume la forma [3, 2, 0, 1, 1] che esprime gli esponenti dei numeri primi (2, 3, 5, 7, 11). Un altro esempio, il numero 980 è rappresentato dal vettore esponente [2,0,1,2]. Un importante proprietà ci dice che la moltiplicazione dei due numeri, nell'esempio 5544 e 980, produce un numero il cui vettore esponente è la somma dei due corrispondenti vettori esponenti, che nel caso specifico sono [3, 2, 0, 1, 1]+ [2,0,1,2,0]= [5,2,1,3,1]. Un'altra importante proprietà derivante da questa rappresentazione ci indica che un numero è un quadrato perfetto se tutte le componenti del vettore esponente sono pari. Così ad esempio [1,2,0] + [3,0,2]= [4,2,2] è il vettore esponente di un quadrato. Per esercizio verifichiamolo. 18 * 200 =3600= 60^2. Le sorprendenti proprietà dei vettori esponenti non finiscono qui. Se facciamo il modulo 2 delle componenti dei vettori e del risultato ottenuto dai vettori

esponenti così modificati e sommati, se il risultato è un vettore nullo allora il numero corrispondente è un quadrato perfetto. Con riferimento all'esempio si ha che [1, 0, 0] + [1, 0, 0] =[0, 0, 0]. Questa caratteristica è particolarmente importante poiché consente di usare metodi di verifica molto veloci come lo XOR bit a bit. A questo punto il problema può essere riformulato come segue: dato un set di vettori nella forma binaria (ottenuti previo modulo 2), trovare un sotto insieme che addizionato dia come risultato un vettore nullo. Si tratta di un problema di algebra lineare e la soluzione è conosciuta come dipendenza lineare. Un modo efficiente per risolvere il problema è utilizzare il metodo di eliminazione di Gauss. Un primo grattacapo che si presenta riguarda la "consistente" grandezza dei vettori associata a numeri anche di "medie" dimensioni, il che produce matrici associate al metodo di Gauss, di grandi dimensioni. Per minimizzare il problema si devono ricercare numeri a tali che a2 mod n abbiano un piccolo numero di fattori primi, si tratta di numeri "adatti" (nella letteratura si indicano come smooth number). Con i numeri adatti la matrice è trattabile. In sostanza l'algoritmo del crivello quadratico può essere per grandi linee sintetizzato nei passi riportati di seguito, più avanti approfondiranno alcuni punti:

1. Scelta di una base di fattori B qualsiasi. Il numero *np(B)* indica il numero di primi che esprime la base; si tenta di controllare sia il numero di vettori che la lunghezza di essi.
2. Usando un crivello individuiamo una base B prodotta da un numero a con a^2 *mod n* tale che B sia una base adatta (smooth).
3. Usando l'algebra lineare troviamo un sotto insieme di questi vettori che addizionati producono il vettore nullo. Quindi la moltiplicazione dei corrispondenti numeri è un quadrato perfetto.
4. Troviamo le due radici quadrate del quadrato modulo n, la prima prendendo la radice quadrata del numero intero, la seconda corrispondente all'originario numero a della base *B* adatta il cui prodotto è il quadrato;
5. Calcoliamo il MCD -massimo comun divisore – (vedi soluzioni di Fabio Grimaldi ioProgrammo n. 99) tra *n* e la differenza (o somma) delle due radici quadrate. Questo produce un fattore che se non banale (n o 1) fornisce la soluzione. Se il fattore è banale si tenta ancora producendo un altro numero a con una nuova dipendenza lineare.

Quindi il crivello quadratico tenta di individuare una coppia di interi *x* e *y(x)*, questo ultimo è una funzione di *x* che soddisfa la congruenza quadratica (1). Si vuole selezionare un insieme di numeri primi, chiamati fattori base B, e tentare di trovare una x tale che il minimo resto della congruenza citata fattorizzi completamente con gli elementi di B. La fattorizzazione di y(x) che si genera sulla base a partire da x è conosciuta come relazione. Il crivello quadratico accelera il suo processo cercando relazioni attraverso la scelta di x chiuse sul quadrato di n. Così siamo assicurati sul fatto che y sia il più piccolo possibile:

$$y(x) = ([\sqrt{n}] + x)^2 - n \quad \text{dove} \quad x \quad \text{è} \quad \text{un} \quad \text{int "piccolo"}$$

$$y(x) \approx 2x[\sqrt{n}]$$

y cresce lineramente con *x* per la radice quadrata di n. Esistono molti modi per verificare che B sia una base adatta. Il più conosciuto è attraverso frazioni continue; in tal modo aumenta significativamente il data collection. Un altro metodo prevede la generazione di curve ellittiche. Di seguito è riportato un processo di cernita, ciò che svolge il crivello.

$y(x) = x^2 - n$
$y(x + kp) = (x + kp)^2 - n$
$y(x + kp) = x^2 + 2xkp + (kp)^2 - n$

$$y(x + kp) = y(x) + 2xkp + (kp)^2 \equiv y(x) \pmod{p}$$

Così risolvendo la congruenza *y(x)=0 (mod p)* per *p* si genera una sequenza totale di ys che sono divisibili per p. Tale risultato è stato trovato con un algoritmo simile a quello proposto da Shanks e Tonelli. Da qui deriva il nome di crivello quadratico. Quadratico proprio perché *y* è un polinomio quadratico in *x*; crivello perché lavora come il crivello di Eratostane.

CONCLUSIONI

Per comprendere appieno l'ultimo metodo proposto, è necessario seguire nella pratica e con esempi come si modifica un'opportuna struttura dati associata all'algoritmo. Inoltre, rimane da vedere il comportamento del metodo in condizioni limite, ad esempio in presenza di un numero primo molto grande. Ed ancora è interessante osservare l'esecuzione passo passo su un problema reale dell'algoritmo. In ogni caso è facile intuire quanto questi metodi abbiano influenzato la teoria della crittografia

Fabio Grimaldi